ARCoSS

LNCS 8283

Leizhen Cai
Siu-Wing Cheng
Tak-Wah Lam (Eds.)

Algorithms and Computation

24th International Symposium, ISAAC 2013
Hong Kong, China, December 2013
Proceedings

Springer

Leizhen Cai Siu-Wing Cheng
Tak-Wah Lam (Eds.)

Algorithms and Computation

24th International Symposium, ISAAC 2013
Hong Kong, China, December 16-18, 2013
Proceedings

Springer

Volume Editors

Leizhen Cai
The Chinese University of Hong Kong
Department of Computer Science and Engineering
Hong Kong, China
E-mail: lcai@cse.cuhk.edu.hk

Siu-Wing Cheng
Hong Kong University of Science and Technology
Department of Computer Science and Engineering
Hong Kong, China
E-mail: scheng@cse.ust.hk

Tak-Wah Lam
University of Hong Kong
Department of Computer Science
Hong Kong, China
E-mail: twlam@cs.hku.hk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-45029-7 e-ISBN 978-3-642-45030-3
DOI 10.1007/978-3-642-45030-3
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013954070

CR Subject Classification (1998): F.2, G.2, I.3, E.1, F.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the proceedings of ISAAC 2013, the 24th International Symposium on Algorithms and Computation, held in Hong Kong, China, December 16–18, 2013. ISAAC is a well-established annual international symposium covering a wide range of topics in theoretical computer science.

ISAAC 2013 received 177 submissions worldwide. Each submission was reviewed by at least three Program Committee members, possibly with the assistance of external reviewers. The Program Committee accepted 67 papers to be presented at the conference, and special issues of *Algorithmica* and *International Journal of Computational Geometry and Applications* are prepared for some selected papers among the accepted ones. The best paper award was given to "RAM-Efficient External Memory Sorting" by Lars Arge and Mikkel Thorup.

Two eminent invited speakers, S. Muthukrishnan of Rutgers University, USA, and Moni Naor of Weizmann Institute of Science, Israel, gave two interesting invited talks at the conference.

> "Cryptography and Data Structures: A Match Made in Heaven," *Moni Naor.*
>
> "Market Approach to Social Ads: The MyLikes Example and Related Problems," *S. Muthukrishnan.*

The talk by Prof. Muthukrishnan was based on a paper included in these proceedings that has the same title.

We are very grateful to all Program Committee members and external reviewers for their excellent work in the difficult selection process. We thank all authors who submitted their work for our consideration. They all contributed to the high quality of the conference. Finally, we thank Conference Co-chairs Chung-Keung Poon and Hing-Fung Ting and all conference volunteers for their dedication that made the conference possible and enjoyable.

December 2013

Leizhen Cai
Siu-Wing Cheng
Tak-Wah Lam

Organization

Program Co-chairs

Leizhen Cai	Chinese University of Hong Kong, Hong Kong, China
Siu-Wing Cheng	HKUST, Hong Kong, China
Tak-Wah Lam	University of Hong Kong, Hong Kong, China

Program Committee

Pankaj Agarwal	Duke University, USA
Hee-Kap Ahn	POSTECH, South Korea
Boris Aronov	Polytechnic Institute of NYU, USA
Therese Biedl	University of Waterloo, Canada
Hubert Chan	University of Hong Kong, Hong Kong, China
Jianer Chen	Texas A&M University, USA
Xi Chen	Columbia University, USA
Pavol Hell	Simon Fraser University, Canada
Wing-Kai Hon	National Tsinghua University, Taiwan
Jesper Jansson	Kyoto University, Japan
Stefan Kratsch	Technische Universität Berlin, Germany
Stefan Langerman	Université Libre de Bruxelles, Belgium
Lap-Chi Lau	Chinese University of Hong Kong, Hong Kong, China
James Lee	University of Washington, USA
Jian Li	Tsinghua University, China
Hsueh-I Lu	National Taiwan University, Taiwan
Julian Mestre	University of Sydney, Australia
Pat Morin	Carleton University, Canada
Benjamin Moseley	Toyota Institute of Technology at Chicago, USA
Evanthia Papadopoulou	Università della Svizerra italiana, Switzerland
Christophe Paul	CNRS, LIRMM, France
Rajeev Raman	University of Leicester, UK
Kunihiko Sadakane	National Institute of Informatics, Japan
Rob van Stee	University of Leicester, UK
Wing-Kin Sung	National University of Singapore, Singapore
Jan Arne Telle	University of Bergen, Norway
Hing-Fung Ting	University of Hong Kong, Hong Kong, China
Antoine Vigneron	KAUST, Saudi Arabia

Lusheng Wang	City University of Hong Kong, Hong Kong, China
Yajun Wang	Microsoft Research Asia, China
Prudence Wong	University of Liverpool, UK
Yitong Yin	Nanjing University, China
Neal Young	University of California, Riverside, USA
Shengyu Zhang	Chinese University of Hong Kong, Hong Kong, China

Conference Co-chairs

Chung-Keung Poon	
Hing-Fung Ting	University of Hong Kong, Hong Kong, China

Additional Reviewers

Abel, Zachary
Aloupis, Greg
Amenta, Nina
Asinowski, Andrei
Aspnes, James
Austrin, Per
Bae, Sang Won
Ballard, Grey
Bannai, Hideo
Barequet, Gill
Bei, Xiaohui
Belazzougui, Djamal
Bhattacharya, Sayan
Boeckenhauer, Hans-Joachim
Bogdanov, Andrej
Bonamy, Marthe
Bose, Prosenjit
Bougeret, Marin
Boyar, Joan
Bringmann, Karl
Buekenhout, Francis
Burcea, Mihai
de Castro, Pedro Machado Manhaes
Chalermsook, Parinya
Cheilaris, Panagiotis
Chen, Fei
Cheung, Ho Yee
Chrobak, Marek
Cormen, Thomas
Devroye, Luc
Dey, Tamal
Dinitz, Michael
Drange, Paal
Duan, Ran
Dujmovic, Vida
Duncan, Christian
Durocher, Stephane
Dósa, György
Elbassioni, Khaled
Etessami, Kousha
Feng, Qilong
Fort, Marta
Fournier, Hervé
Friedrich, Tobias
Friggstad, Zachary
Gagie, Travis
Gambette, Philippe
Gargano, Luisa
Garnero, Valentin
Gawrychowski, Pawel
Gebremedhin, Assefaw
Golovach, Petr
Grandoni, Fabrizio
Guo, Chengwei
Guo, Heng
Gupta, Manoj
Habib, Michel
Harutyunyan, Hovhannes
Haverkort, Herman
Henze, Matthias
van 't Hof, Pim
Horiyama, Takashi
Huang, Zengfeng
Hüffner, Falk
Im, Sungjin
Izumi, Taisuke
Jelínek, Vít
Joret, Gwenaël
Kao, Mong-Jen
Khramtcova, Elena
Kida, Takuya
Kim, Eun Jung
Kim, Hyo-Sil
Kim, Sang-Sub
Kinnersley, Bill
Kiraly, Tamas
Kloks, Ton
Knauer, Christian
Korman, Matias
Koutis, Ioannis
Kowalski, Dariusz
Kratsch, Dieter
Ku, Tsung-Han
Kwok, Tsz Chiu

Lee, Cherung
Lee, Lap-Kei
Leung, Henry C.M.
Leveque, Benjamin
Levin, Asaf
Li, Minming
Li, Shi
Liang, Hongyu
Liao, Kewen
Liedloff, Mathieu
Lin, Tien Ching
Liu, Chi-Man
Liu, Hsiang-Hsuan
Liu, Yang
Liu, Zhenming
Lu, Wei
Luo, Ruibang
Ma, Tengyu
Maddaloni, Alessandro
Maffray, Frédéric
Maheshwari, Anil
Manne, Fredrik
Mans, Bernard
Markou, Euripides
Martin, Russell
Mathieu, Claire
Meolhave, Thomas
Mertens, Stephan
Mondal, Debajyoti
de Montgolfier, Fabien
Morozov, Dmitriy
Mosteiro, Miguel A.
Mozes, Shay
Navarro, Gonzalo
Nederlof, Jesper
Nekrich, Yakov
Ni, Guanqun
Nichterlein, André
Ning, Li
Nonner, Tim
Novotný, Petr
Oh, Eunjin
Okamoto, Yoshio
Ott, Sebastian
Pagh, Rasmus
Pan, Jiangwei
Park, Dongwoo
Poon, Sheung-Hung
Popa, Alexandru
Puglisi, Simon
Radoszewski, Jakub
Raghavendra, Prasad
Ramanujan, M.S.
de Rezende, Pedro
Rutter, Ignaz
Röglin, Heiko
Sach, Benjamin
Sankararamanan,
Swaminathan
Satti, Srinivasa Rao
Saurabh, Saket
Scarcello, Francesco
Schlipf, Lena
Schweitzer, Pascal
Seyffarth, Karen
Shalom, Mordechai
Shin, Chan-Su
Silveira, Rodrigo
Son, Wanbin
Srivastava, Piyush
Streinu, Ileana
Sun, He
Sun, Xiaoming
Sun, Xiaorui
Sviridenko, Maxim
Szeider, Stefan
Szwarcfiter, Jayme
Tang, Bo
Tanigawa, Shin-Ichi
Taslakian, Perouz
Thankachan, Sharma V.
Thorup, Mikkel
Tirthapura, Srikanta
Tiskin, Alexander
Tsang, Hing Yin
Uehara, Ryuhei
Uno, Yushi
Vatshelle, Martin
Vigoda, Eric
Vilcu, Costin
Vredeveld, Tjark
Wang, Haitao
Wei, Zhewei
Weller, Mathias
Wong, Sam Chiu-Wai
Wood, David R.
Wu, Will
Wu, Xiaowei
Wu, Yaokun
Xia, Ge
Xiang, Xiangzhong
Xiao, Mingyu
Yamanaka, Katsuhisa
Yang, Shenghao
Yoon, Sang Duk
Yu, Albert
Yu, Hung-I
Yu, Sheng
Yuster, Raphael
Zavershyskyi, Maksym
Zhang, Chihao
Zhang, Jialin
Zhang, Qin
Zhang, Wuzhou
Zhang, Yong
Zhao, Zhichao
Zhou, Bin
Zhou, Hai
Zhou, Yuan
Zhu, Zeyuan Allen

Table of Contents

Session 10B: Network Algorithms

Market Approach to Social Ads: The MyLikes Example and Related Problems

Darja Krushevskaja[1] and S. Muthukrishnan[2]

[1] Rutgers University
[2] Rutgers University & Microsoft Research

Abstract. A potential way to advertise on social networks is to rely on word of mouth. What is a market approach to word of mouth social advertising? We describe an example: MyLikes, which is a new advertising platform. It lets anyone on a social network be a "publisher" of advertisements (ads). It provides a matching market so advertisers can find social publishers to advertise their products. Further, interestingly, it lets the social publishers modify the ad. As a result, a single base ad may be morphed into many. MyLikes lets publishers broadcast and communicate the ads to others in their social network. Finally, it provides a mechanism for advertisers to pay the publishers based on engagement of the social users with the ads. We describe research problems that emerge in such a marketplace for social ads.

1 Introduction

There are established online ad markets like sponsored search and display ads. In sponsored search, users are shown ads in response to their searches and based on variety of signals including the searches, history and location of the users. In display ads, users are shown ads in response to their browsing behavior based on signals including visited websites, behavioral profiles and so on. These established ad markets have led to a lucrative and successful online ad industry.

Our focus is on ads in social platforms. Obviously, social platforms like Facebook, Twitter and others provide new signals for ads targeting, such as ones' friends and followers, topics of interest, events in their life, likes, and so on. These signals seem rich and promising, and so they could be immediately used within established ad markets like display ads for social platforms. In addition, social platforms provide other opportunities for ads, unique to their platform, such as word of mouth. Social platforms let people talk to a large number of others — grouped according to social circles or interests — directly. So, instead of getting global celebrities to endorse a product and using broadcast systems like TV to reach many users, social networks offer an alternative, which is to use several social entities to each reach out to islands of users.

We focus on such word of mouth social ads. There has been prior research focus on individual problems like finding "influential" social entities, and there is fledgling industry around providing social entities with products so they can plug them on social networks. We go beyond these and address the central question of what would be an overall market that will enable social ads via word of mouth.

- We describe an example of a marketplace for word of mouth social ads, namely, MyLikes.
- We present research questions of interest from optimization to machine learning and strategic problems that arise.

We hope to initiate research on market approaches to social ads.

2 Example: The MyLikes Platform

MyLikes is a word of mouth advertising platform. It has a direct recruiting model for advertisers to find publishers via a market, who will publish their ads to their social circle. It works on top of certain existing social mediums, such as Twitter or YouTube. We focus on Twitter-based MyLikes. At high level, the market works as follows.

- *Parties Register with MyLikes.* Any registered social user of Twitter can register with MyLikes to be a *publisher* p. Any advertiser a who wishes to advertise their product through the word of mouth may register at MyLikes and set up an ad campaign C.
- *Matching of the Two Parties.* MyLikes matches publishers p's to campaigns C's using multiple signals and presents each p with a ranked list L of a subset of campaigns. p's choose from their lists.
- *Word of Mouth Advertising.* p may modify the creative (ad) in the campaign C they choose and create a *sponsored post* $p(C)$ with the ad. MyLikes tweets $p(C)$ on behalf of p. Tweet $p(C)$ will appear on timeline of publisher p along with her other posts.
- *Ad Engagement and Pricing.* Any follower $f(p)$ of p may interact with the $p(C)$, clicking or retweeting $p(C)$ and so on. MyLikes determines a price for the engagement, charges the advertiser a and pays the publisher p, after taking out their cut.

There are a number of crucial details behind this high level picture that ultimately determine the precise market. Examples include, how do advertisers a's specify their target users for campaign C's, how is this targeting used in the matching and pricing steps, how is the engagement different for sponsored vs standard posts, how do advertisers control their ad intent despite the modifications by publishers, and many others. Figure 2 contains the overview of the market. In what follows, we will describe these details.

Registration at MyLikes. Anyone with a Twitter account can become a publisher p on MyLikes platform. First, p creates a MyLikes account. Then, she needs to authorize MyLikes *application* to use her Twitter account. To do so, p has to click on a button placed on main page of her MyLikes account, login into Twitter

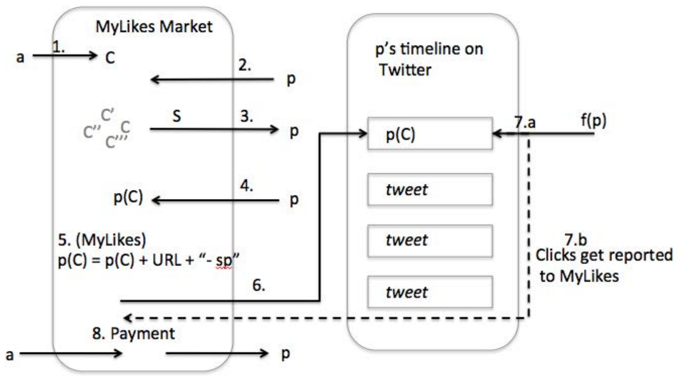


Fig. 1. The timing of MyLikes market for social ads

with her Twitter credentials. This will "attach" her Twitter account to MyLikes account. Starting from that moment, MyLikes will be able to publish on behalf of the user. MyLikes is also able to read tweets from p's timeline, collect information on the followers $f(p)$ of p, and post tweets on behalf of p.

Any advertiser can register with MyLikes. Say advertiser a has a target audience $G(a)$ of Twitter users they wish to reach. They set up a campaign C:

– *Metadata:* title, description and thumbnail
– *Creative:* language to use to address audience and sample posts (optional)
– *Target:* G(a). At this moment, $G(a)$ can be only determined by geolocation of users and device type (mobile or stationary), but one can posit other languages to specify the properties of target users.
– *Engagement:* landing page URL
– *Payment:* weekly budget B and per click pay b.

Advertiser agrees to pay *at most* b for each *eligible* click. The click is *eligible* if it satisfies targeting criteria, that is, user who clicked falls in $G(a)$. The payment for a click may be smaller than b and the total weekly payment of the advertiser a does not exceed specified weekly budget of the campaign B. The MyLikes platform provides advertisers with a set of tools that allows them to maintain, modify and monitor campaigns.

The Matching. Say a publisher p accesses MyLikes. This is a recruitment opportunity. MyLikes considers the set of all available p's, set of all available ad campaigns C's, and together with all the signals including target properties of campaigns, behavior of p's and their followers, and other signals in real time. Then, p is presented a suitable subset of campaigns as ordered lists L's. Typical ordering included ordering by revelance, newness, follower click rate, price per click/view etc. Clearly the processing MyLikes performs to determine L's is crucial to their business. This has to be strategic, balancing short term and long term revenues of the market, with a good understanding of the various strategies of the publishers and advertisers, even while modeling and understanding the behavior of users. In the list, each campaign has a thumbnail, title, description, targeting criteria and estimated pay-per-click. Publisher p may choose any number of campaigns from the list.

Word of Mouth Advertising. Say publisher p chooses an ad campaign C. p is shown a (possibly empty) list of sample posts from advertiser a for campaign C. p can choose any of these sample posts, can morph them to her liking or can create her own version of the post. Once satisfied, publisher p submits it to MyLikes. MyLikes creates a special url for each post of p which first directs user who clicked on it to MyLikes servers, and then to the URL specified by a in the description of C. Then MyLikes appends generated url and "- sp" tag to the post submitted by p and creates the composite sponsored post $p(C)$, and posts it on Twitter on behalf of p. Tweet $p(C)$ will appear on timeline of publisher p along with her other posts and is visible to the followers $f(p)$ of p.

MyLikes registers a short URL at `bitly.com` for each sponsored post. A click on ad C's short URL is first sent to Twitter which redirects it to `bitly.com`, which in turn directs user to MyLikes servers. From MyLikes servers, user is redirected to C's landing page. Detailed are in Figure 2.

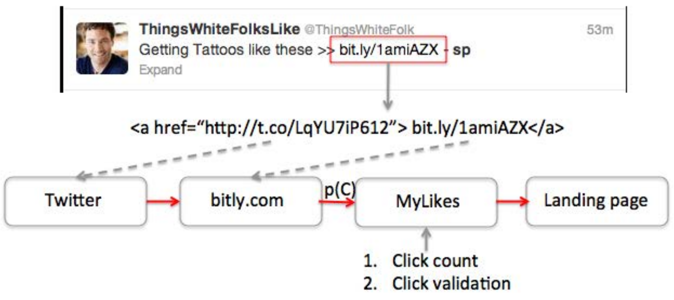

ThingsWhiteFolksLike @ThingsWhiteFolk
53m
Getting Tattoos like these >> bit.ly/1amiAZX - sp
Expand
<a href="http://t.co/LqYU7iP612"> bit.ly/1amiAZX</a>
Twitter
bitly.com
p(C)
MyLikes
Landing page
1. Click count
2. Click validation

Fig. 2. 3-hop redirect system used by MyLikes for click accounting

Engagement and Pricing. Users see $p(C)$ on the timeline of Twitterers they follow. Users may not be able to tell the difference between a sponsored vs standard non-sponsored post visually, unless they know the meaning of the suffix "- sp" and are perspicacious and see it. As a result, thus far, MyLikes can work potentially independent of Twitter without any explicit help or support from it. In some cases publisher decides and deliberately removes the suffix: in order to get paid publisher only needs to keep the URL from the post intact. There are a variety of ways for the publisher to do this in the platform and likewise, a variety of ways for MyLikes to detect that and deter such behavior.

Users may engage with $p(C)$ in many ways. For example, a follower $f(p)$ of p may click on the tweet $p(C)$. She may also retweet it, and some follower $f(f(p))$ may click on $p(C)$, and so on. When some user clicks on $p(C)$, MyLikes can track the event via their URL for accounting purposes, and forward the click to the campaign URL.

Pricing depends on the user u who clicked on $p(C)$. Since MyLikes can track clicks, MyLikes can determine (or estimate) if u meets targeting criteria $G(a)$. If yes, MyLikes determines a payment $q \leq b$ of C and the amount is paid to publisher p. This payment q may vary over time for the same p and C, may depend on u and p, or auction involved in the list of campaigns shown to p. Similarly to advertisers, MyLikes provides a dashboard for publishers to follow performance of their sponsored posts.

2.1 Observations on the Market

Here are some observations on the market for social ads, such as MyLikes.

- Advertisers specify the target users they wish to reach, and not the intermediate publishers they wish to have advertise their products. This is an interesting decision that removes the burden of advertisers thinking about "how" to reach their audience, and instead shifts the burden to the platform to find the most effective matching of publishers that will result in reaching the target audience.
- Publishers can rewrite creatives or write a new creative. This is a significant departure in the ads industry where brands control the creative. The insight is that each publisher knows their audience and crafts or morphs the ads to be suitable for their followers. It will be difficult for the advertiser to achieve this effect on their own, of thousands of smart, incentivized intermediaries writing versions of ads suitable to islands of users. This can be thought of as crowd sourcing the creation of ads.
- One approach to do word of mouth dissemination is to mine the data to identify the "influential" entities. But in the market approach, no particular model of influence is baked in, either in campaigns or in the platform. Empirical and predicted effectiveness is used to focus publishers' attention on a list L of ads, but each publisher gets to pick their campaigns. Being selfish agents, they will strategize in ways that may not fit any succinct model of influence. For instance, a publisher may let a lucrative ad campaign go if they thought their followers have seen a lot of ads recently or if their followers will likely overlap with a different publisher that they think will pick up this campaign etc. Such strategic behavior from each potential publisher will determine the ultimate success of this market.
- In any word of mouth advertising platform, one can not guarantee that ad reaches *only* the target audience. Each intermediary or publisher p reaches that audience, some of whom will match $G(a)$ and others will not. MyLikes design gives the platform the access to followers information so that from their profile and behavior, MyLikes can estimate the part of the audience that does not satisfy $G(a)$.

- The pricing model is simple. A publisher p who posts a sponsored post gets paid not only for all the users who hear from them *directly* and click on the sponsored post, but also others who do that but due to retweeted posts, even recursively. While one can debate, research and try out different schemes for compensating the intermediate publishers in chains that led to users' clicking on a sponsored post, the basic design choice is elegant and interesting.
- Unlike sponsored search and display ads where the platform can control the number of impressions (and also the number of clicks) of an ad and hence guarantee not exceeding the budget, the social platform has an inherent challenge. Once a campaign begins as a sponsored post, the post may get clicked on any number of times, possibly even after retweeting. So, if one charges per click, at some point, the budget will be exhausted and the campaign can continue to generate impressions and clicks, *without* getting charged.
- There are certain challenges. For example, what if an advertiser finds one of the publishers of their ad to be unsuitable or finds the sponsored post unsuitable to their brand. This can be controlled by quickly detecting and eliminating offending creatives and publishers. Also, this social market, by working independent of Twitter, is unable to track the number of impressions of an ad. Help from Twitter will help address these challenges.

3 Overview of Novel Research Directions

In what follows, we provide an overview of research problems that arise in word of mouth social ad markets.

Strategy Space. Consider an advertiser a who has a true underlying value v per user engagement, true budget B^* in mind and true target $G(a)^*$. Advertiser has many strategies available to them, reporting a bid b, budget B and target $G(a)$ — all possibly different from the true quantities — and experimenting with multiple campaigns. By misreporting these parameters, the advertiser might benefit. For example:

- Say a misreports $G(a)$ to be $G'(a)$ and for the same budget, reach some from $G'(a)$ but also a large number of those in $G(a)$ for no additional cost as a consequence just because of the overlap of $G(a)$ and $G'(a)$ in the broadcast communication platform.
- Say a reports B lower than B^*. Publishers p may still be tempted, but p will exhaust the budget soon. The tweet will still remain in the system and will be clicked on and retweeted by many, for which a will not have to pay.
- Say p chooses a campaign C to sponsor. p can modify it to tune the engagement of any subset of its followers to C.

Hence, underlying mechanisms in social ad markets have to contend with such strategies.

Learning and Estimation. There are many parameters that need to be learned. This includes

- For p and $G(a)$, the estimated number of impressions, clicks, retweets and other engagements. p's need these to figure out if they should choose to tweet a campaign and a's need these to design their campaigns.
- For each p and campaign C of some advertiser a, the *rate* at which the above quantities accumulate. Since user engagement happens potentially in the future, the platform needs these estimates to match ads to publishers based on their potential leftover budget.

These items have be to learned as statistical quantities without regard to the strategies of players, or as e.g. regret learning, as online estimations in presence of strategic agents. Further, feedback from users such as click on retweets arrives delayed over time, and hence exploration and estimation has to work in large time scales.

Auction/Allocation/Optimization Problems. There are three major players: advertiser, publisher and the market platform.

- The publisher has to determine the series of ads to promote and optimize both short and long term revenue and satisfaction of their followers.
- The advertiser has to consider v, B^* and $G^*(a)$'s and construct ad campaigns in order to maximize their profit or brand goals, while cognizant of the other advertisers in the market.
- The platform needs to select a small set of a's to show to p, rank them into an ordered list, determine a pricing for user engagement on each, and so on.

These problems have been studied extensively in sponsored search and display ads markets, and studying them in markets for social ads leads to new research problems.

Consider a publisher p. Assume that each follower $f(p)$ is a node in a bipartite graph, as is each potential campaign C. We will use $G(C)$ as the target audience set of C, using it to denote both the set of features of the users and the set of users with those features. Then, there is an edge $(f(p), C)$ if $f(p) \in G(C)$. We can also assume that for each edge $(f(p), C)$, p knows the quality of engagement. This defines a weighted bipartite graph, similar in spirit to sponsored search and display ads. Now we have a few departures. C's arrive online or can be assumed to be may be known *a priori* for benchmarking purposes as is standard in those markets. In social ad markets, since a publisher may tweet more than once for any campaign, we also have an intermediate model where the campaigns that have arrived thus far are known and persistent, but the others are online. A bigger departure is that p may morph the campaign ads strategically to tune the quality of edge $(f(p), C)$ in a joint way among groups of $f(p)$'s. Another big departure is that p has to solve a *production* problem. Knowing the arrival estimates of C's, the tweet consumption habits of their $f(p)$'s and the graph above, at what (say periodic) rate should p tweet sponsored posts, and during each tweeting, which and how many sponsored posts should p tweet and how should they be morphed and targeted? One can formulate a suitable objective function and solve the offline or online versions. In contrast, sponsored search and display ads markets typically study yield optimization which for a given inventory determines which ads to place or procure, and exogenizes the production of inventory [5,1].

Consider an advertiser a with true budget B^*, target audience $G^*(a)$, and true value v per engagement v that is identical for all engagements from the target. a needs to set up campaigns $C_1, \ldots, C_k$ for some k, each with its budget B_i, bid b_i and target $G(C_i)$ such that the total budget is no more than B, and the engagement from $G^*(a)$ is optimized. This is akin to *budget optimization* in sponsored search and display ads markets [4]. At the high level, one commonly approaches this as a best response problem given summary of the impact of other campaigns in the market, which ultimately becomes knapsack problems given estimates of performance of each campaign C_i. In social ad markets, the departure happens when one wishes to optimize not the total impressions or clicks for $\sum_i |G(C_i)|$, but rather say their union, $|\sum_i G(C_i)|$. Another departure occurs when a considers *negative* audience. Each campaign C_i with target $G(C_i)$ reach an audience $T(C_i)$ that overlaps with $G(C_i)$ but also contains others and generates engagement from an unintended audience. It is reasonable to model a negative audience $N(C_i) \in T(C_i)$, $N(C_i) \cap G(C_i) = \phi$, and a then needs to optimize over both $G(C_i)$ and penalize for $N(C_i)$'s. This leads to potentially hard variants of broad matching problems in sponsored search [3].

Consider the market. The market has a bipartite graph as well. Each publisher p is a node in the graph, as is each ad campaign C. For each edge (p, C), the market knows the engagement numbers for target $G(C)$ among the followers $f(p)$ of p. The problem of determining which campaigns to present to the p's with both short term and long term revenues in mind is an ad allocation problem. There is a nice literature on ad allocation problems as they arise in sponsored search and display ads, and their formulation as online matching problems with possibly sub modular welfare functions [7,8]. In social ad markets, both C's and p's arrive online, C's persist and p's may arrive several times, and this departs from the online arrival of perishable impressions in sponsored search and display ads markets. Another departure is that the overall metric optimize may not be a function of the total engagement from each p, but in terms of the union of users in $f(p)$'s which induces set union based objectives. Also, each occurrence of p is related to other occurrences, and this induces graph constraints on one of the partitions. Most crucially, ad allocation has to be solved in presence of strategies of p's. This does not have an immediate analogy in ad allocation problems in sponsored search and display ads where users (impressions) are not thought to be strategic.

One approach to ad allocation is repeated auctioning with some budget admission control on top. Then the question is what auctions are suitable. It is natural to assume that when a publisher arrives, all applicable campaigns can be considered and some version of Generalized Second Price (GSP) auction may be run, as in sponsored search [2,13]. The departure in social ad markets is that

p chooses a campaign strategically, in contrast to sponsored search where users click on ads based on their utility to them. This leads to new modeling and formulation of the auctioning problem.

Each of the above directions can be abstracted into concrete problems that are open in theory.

4 Related Work

There is considerable literature on ad campaigns in sponsored search and display ads, and in particular on the use of keywords, websites and user profiles in these platforms [2,13,5,11], as well as real time exchanges. Some open research problems are in [9,10].

In social networks, display advertising is prevalent. However, ads are shown to users using additional signals drawn from the social network, e.g., connections, interests, etc.. Such additional information is believed to be valuable `http://techcrunch.com/2013/06/11/salesforce-facebook-ads-benchmark`.

Our focus is on word of mouth marketing. There is an industry that provides products to certain individuals and lets them promote the product. An underlying problem is finding the influential people to target. There are number of models for influence and finding sets of influential people in these models is often hard [12,6].

Instead we focus on market methods. Besides Mylikes, there are other examples, such as `http://ad.ly/`, `http://sponsoredtweets.com/`, and `http://revtwt.com/`.

5 Concluding Remarks

We described a market approach to social ads, and the broad class of research problems that arise. We believe this will be an interesting research agenda for the community. Exending this approach to work over other social platforms such as YouTube and Facebook brings up other interesting issues too.

References

1. Balseiro, S., Feldman, J., Mirrokni, V., Muthukrishnan, S.: Yield optimization of display advertising with ad exchange. In: Proceedings of the 12th ACM Conference on Electronic Commerce, pp. 27–28. ACM (2011)
2. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Technical report, National Bureau of Economic Research (2005)
3. Even-Dar, E., Mirrokni, V., Muthukrishnan, S., Mansour, Y., Nadav, U.: Bid optimization for broad match ad auctions. In: WWW, pp. 231–240 (2009)
4. Feldman, J., Muthukrishnan, S.: Algorithmic methods for sponsored search advertising. In: Performance Modeling and Engineering, pp. 91–124 (2008)

5. Ghosh, A., McAfee, P., Papineni, K., Vassilvitskii, S.: Bidding for representative allocations for display advertising. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 208–219. Springer, Heidelberg (2009)
6. Kempe, D., Kleinberg, J., Tardos, É.: Influential nodes in a diffusion model for social networks. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1127–1138. Springer, Heidelberg (2005)
7. Mehta, A.: Online matching and adwords, `http://www.cs.princeton.edu/~zdvir/apx11slides/mehta-slides.pdf`
8. Mirrokni, V.: Online ad allocation, and online submodular welfare maximization, `https://smartech.gatech.edu/bitstream/handle/1853/43305/Mirrokni.pdf`
9. Muthukrishnan, S.: Internet ad auctions: Insights and directions. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 14–23. Springer, Heidelberg (2008)
10. Muthukrishnan, S.: Ad exchanges: Research issues. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 1–12. Springer, Heidelberg (2009)
11. Riederer, C., Erramilli, V., Chaintreau, A., Krishnamurthy, B., Rodriguez, P.: For sale: your data: by: you. In: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, p. 13. ACM (2011)
12. Singer, Y.: How to win friends and influence people, truthfully: Influence maximization mechanisms for social networks. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 733–742. ACM (2012)
13. Varian, H.R.: Position auctions. International Journal of Industrial Organization 25(6), 1163–1178 (2007)

Geodesic-Preserving Polygon Simplification[⋆]

Oswin Aichholzer[1], Thomas Hackl[1], Matias Korman[2], Alexander Pilz[1], and Birgit Vogtenhuber[1]

[1] Institute for Software Technology, Graz University of Technology, Austria
{oaich,thackl,apilz,bvogt}@ist.tugraz.at

[2] Dept. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain
matias.korman@upc.edu

Abstract. Polygons are a paramount data structure in computational geometry. While the complexity of many algorithms on simple polygons or polygons with holes depends on the size of the input polygon, the intrinsic complexity of the problems these algorithms solve is often related to the reflex vertices of the polygon. In this paper, we give an easy-to-describe linear-time method to replace an input polygon $\mathcal{P}$ by a polygon $\mathcal{P}'$ such that (1) $\mathcal{P}'$ contains $\mathcal{P}$, (2) $\mathcal{P}'$ has its reflex vertices at the same positions as $\mathcal{P}$, and (3) the number of vertices of $\mathcal{P}'$ is linear in the number of reflex vertices. Since the solutions of numerous problems on polygons (including shortest paths, geodesic hulls, separating point sets, and Voronoi diagrams) are equivalent for both $\mathcal{P}$ and $\mathcal{P}'$, our algorithm can be used as a preprocessing step for several algorithms and makes their running time dependent on the number of reflex vertices rather than on the size of $\mathcal{P}$.

1 Introduction

A *simple polygon* is a closed connected domain in the plane that is bounded by a sequence of straight line segments (edges) such that any two edges may intersect only in their endpoints (vertices), and such that in every vertex exactly two edges intersect. Let $\mathcal{P}$ be a simple polygon and let $\mathcal{H}_1, \ldots, \mathcal{H}_k$ be a set of pairwise-disjoint simple polygons such that $\mathcal{H}_i$ is contained in the interior of $\mathcal{P}$ for $1 \leq i \leq k$. Then the closure $\mathcal{Q}$ of $\mathcal{P} \setminus \bigcup_{1 \leq i \leq k} \mathcal{H}_i$ is called a *polygon with holes*. The polygons $\mathcal{H}_i$ are called the *holes* of $\mathcal{Q}$, and the vertices and edges of $\mathcal{Q}$ are the vertices and edges of $\mathcal{P}$ and all $\mathcal{H}_i$, respectively. We regard $\mathcal{P}$ and $\mathcal{Q}$ as closed sets, i.e., they include their boundary. All polygons we consider are

⋆ Research supported by the ESF EUROCORES programme EuroGIGA - ComPoSe, Austrian Science Fund (FWF): I 648-N18 and grant EUI-EURC-2011-4306. T.H. supported by the Austrian Science Fund (FWF): P23629-N18 'Combinatorial Problems on Geometric Graphs'. M.K. received support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union. A.P. is recipient of a DOC-fellowship of the Austrian Academy of Sciences at the Institute for Software Technology, Graz University of Technology, Austria.

either simple polygons or polygons with holes. A vertex of a polygon is reflex if its inner angle is larger than 180°. Given a polygon $\mathcal{P}$ with n vertices, the *geodesic path* $\pi_{\mathcal{P}}(p, q)$ between two points p and q of $\mathcal{P}$ is defined as the shortest path that connects p and q among all the paths that stay within $\mathcal{P}$. The length of that path is called the *geodesic distance*. For any pair of points in $\mathcal{P}$, such a path always exists and, when $\mathcal{P}$ is simple, is unique. Moreover, such a path is a polygonal chain whose vertices (other than p and q) are reflex vertices of $\mathcal{P}$. When the path $\pi_{\mathcal{P}}(p, q)$ is a straight line segment, we say that p *sees* q (and vice versa).

We say that two polygons $\mathcal{P}$ and $\mathcal{P}'$ have the same reflex vertices if for any reflex vertex $v \in \mathcal{P}$ there is a reflex vertex $v' \in \mathcal{P}'$ at the same point in the plane and vice versa. We say that $\mathcal{P}'$ *subsumes* $\mathcal{P}$ if $\mathcal{P}$ and $\mathcal{P}'$ have the same reflex vertices and $P \subseteq P'$. See Fig. 1 for examples.

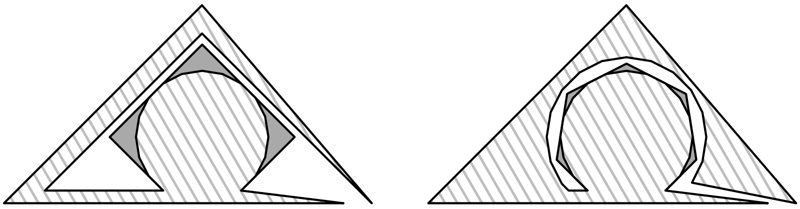


Fig. 1. Simple polygons (hatched) drawn on top of subsuming polygons (solid)

Observation 1. *Let $\mathcal{P}, \mathcal{P}'$ be two simple polygons such that $\mathcal{P}'$ subsumes $\mathcal{P}$. Then, for any $p, q \in \mathcal{P}$ we have $\pi_{\mathcal{P}}(p, q) = \pi_{\mathcal{P}'}(p, q)$.*

For algorithms that solely rely on geodesic paths inside $\mathcal{P}$, the output will remain equivalent if we replace $\mathcal{P}$ by $\mathcal{P}'$ in the input. It is therefore desirable to construct a subsuming polygon $\mathcal{P}'$ such that (1) $\mathcal{P}'$ has few vertices and (2) the construction can be done in time linear in the size of $\mathcal{P}$. For the creation of a subsuming polygon of minimal size, one has to connect the reflex vertices by pairwise non-intersecting paths in the union of the exterior and the boundary of the original polygon. Guibas, Hershberger, Mitchell, and Snoeyink [14] address various aspects of the problem of approximating polygonal paths and polygons by simpler ones. They show that the problem of finding a minimum link simple polygon (of a given homotopy class) having its boundary inside a given region R is NP-hard. In this paper, we show the following result.

Theorem 1. *For any polygon $\mathcal{P}$ (possibly with holes) of n vertices out of which $r > 0$ are reflex, there exists a polygon $\mathcal{P}'$ with $O(r)$ vertices that subsumes $\mathcal{P}$. Moreover, $\mathcal{P}'$ can be computed in $O(n)$ time and will have the same number of holes as $\mathcal{P}$.*

Before giving the proof of Theorem 1 in Section 3, we review some basic properties of pointed pseudo-triangulations (or *geodesic triangulations*) in Section 2.

Section 4 gives an overview of the various applications of our result. In Section 5 we summarize our approach and give a short account on the open problem of computing an optimal subsuming polygon. Due to space constraints, the proofs of some claims are deferred to the full version of the paper [1].

Related Work

We follow the common aim of relating the time and space complexity of algorithms on polygons not only to the input size, but also to the number of reflex vertices. This is often a more significant parameter for the actual difficulty of the problem instance. Hertel and Mehlhorn [18] give an algorithm for triangulating a simple polygon of n vertices, r of which are reflex, in $O(n \log r)$ time. Bose et al. [8] give a $\Theta(m + n \log r)$ algorithm for computing a geodesic ham-sandwich cut of two given sets of m points in a simple polygon with r reflex vertices.

A related way of giving a more fine-grained analysis of algorithms on polygons is by expressing its complexity in the number of edges in the visibility graph (i.e., the number of point pairs that see each other), see for example [11, p. 68]. Note that, e.g., for computing the minimum weight triangulation of a simple polygon, the currently known worst case appears when there are no reflex vertices [7].

The term "polygon simplification" is also used in connection with operations that are used to compress polygons and to reduce noise in the representation. A well-known algorithm to smoothen polygonal chains is the Douglas-Peucker algorithm [10]. Guibas et al. [14] address several variations of the approach to fatten existing polygonal chains and approximate the chain inside the fattened region. There also exists work on constructing simple polygons that contain given ones and fulfill certain properties, as a generalization of the convex hull of simple polygons [4]; the main objective there is to approximate the shape.

2 Pseudo-Triangulations

In this section, we recall several properties of pseudo-triangles and pointed pseudo-triangulations of simple polygons. For details on pseudo-triangulations in various contexts see the survey by Rote, Santos, and Streinu [24].

A *pseudo-triangle* is a simple polygon with exactly three convex vertices. The three convex vertices are the *corners* of the pseudo-triangle, and the three polygonal chains between the corners are called the *side chains*. Note that a side chain might consist only of one edge.

A *pseudo-triangulation* of a simple polygon P is a partition of P into pseudo-triangles, such that the union of the vertices of the pseudo-triangles is exactly the vertex set of P. A vertex v of a pseudo-triangulation is called *pointed* if it is incident to a face in which the angle at v is larger than 180°. In a *pointed pseudo-triangulation* every vertex is pointed. Throughout this work, we are only concerned with pointed pseudo-triangulations. It can be shown that a pointed pseudo-triangulation of a simple polygon with c convex vertices has $c-2$ pseudo-triangles and adds $c-3$ diagonals to the polygon (see, e.g., [24]). Our main result heavily relies on that fact. Guibas et al. [13] showed that, given a vertex v of a triangulated simple polygon P, the set of all shortest paths between v and the vertices of P (i.e., the shortest path tree of v) can be constructed in linear time (this algorithm was later simplified by Hershberger and Snoeyink [17]). The union of all these shortest paths gives a pointed pseudo-triangulation of P. Hence, given a triangulation of a simple polygon P, a pointed pseudo-triangulation of P can be constructed in linear time.[1]

Let ∇ be a pseudo-triangle. The line ℓ bisecting the angle at any corner of ∇ separates the two adjacent side chains C_1 and C_2, and will leave ∇ through the third chain C_3. The bisecting line ℓ' of the angle of a second corner, say, the one joining C_1 and C_3, separates C_1 and C_3. Therefore, ℓ and ℓ' intersect inside ∇ and hence separate C_1 from C_2 and C_3. This allows us to make the following basic observation.

Observation 2. *For any side chain C of a pseudo-triangle ∇, the bisectors of the angles at the corners of C define a wedge that separates C from the remaining boundary of ∇.*

We call this wedge the *separating wedge* of the side chain. Observation 2 is the crucial property of pseudo-triangles that we will use in the next section.

3 Proof of the Main Theorem

We call a simple polygonal chain C *hull-honest* if it is completely contained in the boundary of its convex hull $\mathcal{CH}(C)$. We call a simple chain $C = \langle v_1, v_2, \ldots, v_k\rangle$ *simplifiable*, if it is hull-honest and if the ray $r(v_1, v_2)$ (i.e., the ray from v_1 through v_2) and the ray $r(v_k, v_{k-1})$ intersect. See Fig. 2. Note that if a chain is simplifiable, then any of its subchains is simplifiable as well. For any simplifiable chain C of vertices $v_1, v_2, \ldots v_k$ we introduce an operation called the *simplification* of C as follows: If $k \leq 3$ then C remains unchanged. Otherwise, consider the rays $r(v_1, v_2)$ and $r(v_k, v_{k-1})$. As $k > 3$, these two rays intersect at a point m not on the chain. In that case, we replace C by the chain $C' = \langle v_1, m, v_k\rangle$.

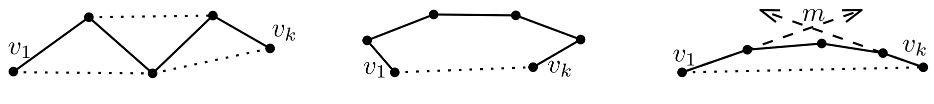


Fig. 2. A chain that is not hull-honest (left), a hull-honest chain (center), and a simplifiable chain (right)

The basic idea of our construction is to simplify long convex chains along the boundary of $\mathcal{P}$. The main challenge is to avoid that the edges introduced by the simplification intersect with the remaining boundary of the resulting polygon.

[1] The connection between pointed pseudo-triangulations and the shortest path tree was mentioned by Speckmann and Tóth [25]; the concept of pseudo-triangulations has been developed after the writing of [13].

Lemma 1. *All the side chains of any pseudo-triangle are simplifiable. When simplifying an arbitrary number of pairwise interior-disjoint subchains, the resulting polygon is again a pseudo-triangle, and, in particular, its boundary is not self-intersecting.*

Using this result we can now prove our main theorem.

Proof (Theorem 1). We first consider the case where $\mathcal{P}$ is a simple polygon with at least one reflex vertex. Let $\mathcal{CH}(\mathcal{P})$ be its convex hull (which can be computed in linear time using, e.g., Melkman's algorithm [19]). The set $\mathcal{CH}(\mathcal{P}) \setminus \mathcal{P}$ is the union of (the interiors of) simple polygons whose interiors are pairwise disjoint. We call these polygons the *pockets*. Each pocket P_i is defined by exactly one convex hull edge, which is not part of $\mathcal{P}$ and is called a *lid edge*, and a subchain C of the boundary of $\mathcal{P}$. The c_i convex vertices of a pocket therefore consist of the reflex vertices of $\mathcal{P}$ along C and the two vertices of the lid edge. Thus, a pointed pseudo-triangulation of P_i has $c_i - 3$ diagonals. We call the lid edges and the pseudo-triangulation diagonals the *support edges*. For p pockets, the number of support edges is $p + \sum_{i=1}^{p}(c_i - 3) = p + r + 2p - 3p = r$. Since the only vertices possibly shared by two pockets are the convex hull vertices, we can construct a pointed pseudo-triangulation of each pocket in accumulated $O(n)$ time for all pockets. See Fig. 3 for an example of a pseudo-triangulated pocket.

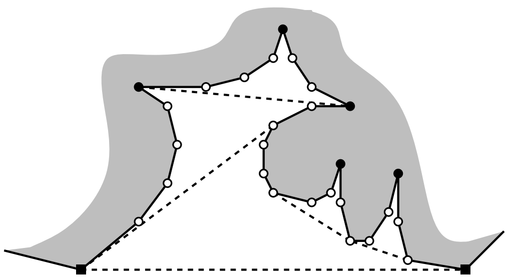


Fig. 3. A pocket and its pointed pseudo-triangulation. Support edges are drawn dashed.

Consider a pocket P_i and a pointed pseudo-triangulation $T(P_i)$ of P_i. A side chain of a pseudo-triangle of $T(P_i)$ consists of convex chains of $\mathcal{P}$, possibly separated by support edges. Note that any vertex of $\mathcal{P}$ that is not on $\mathcal{CH}(\mathcal{P})$ is clearly part of a side chain of at least one pseudo-triangle of $T(P_i)$ in some pocket P_i of $\mathcal{P}$. For each pseudo-triangle in every pocket of $\mathcal{P}$ we simplify all maximal subchains of its side chains that do not contain support edges. Due to Lemma 1, the resulting polygon is again simple and subsumes $\mathcal{P}$.

Observe that $\mathcal{CH}(\mathcal{P})$ consists of convex chains of $\mathcal{P}$, possibly separated by lid edges. Unlike the maximal convex chains inside pseudo-triangles, there might exist a maximal convex chain (not containing lid edges) $C = \langle v_i, \ldots, v_j \rangle$ on $\mathcal{CH}(\mathcal{P})$ that is not simplifiable. The reason for this being, that the turn of the chain is at least 180°, i.e., the rays $r(v_i, v_{i+1})$ and $r(v_j, v_{j-1})$ do not intersect. However, such a chain C appears at most once on $\mathcal{P}$ and can be split into two simplifiable parts with a common vertex v^*.

It remains to count the number of convex vertices of the resulting polygon $\mathcal{P}'$. To this end, we charge the convex vertices of $\mathcal{P}'$ either to reflex vertices or to one of the r support edges. Let $C = \langle v_i, \ldots, v_j \rangle$ be a maximal convex chain not containing support edges of $\mathcal{P}'$. Each of the two end points of C is either a reflex vertex of $\mathcal{P}$ (or v^*), or end point of a support edge, or both. If v_i is the end point of a support edge e, then we charge C to e. Otherwise, we charge C to the reflex point v_i (or to the special point v^*). Observe that each end point of a support edge can be at most once a starting point (v_i) of such a maximal convex chain C of $\mathcal{P}'$. The same is true for each reflex vertex of $\mathcal{P}$ (and v^*). Thus, $\mathcal{P}'$ consists of at most $2r + r + 1$ maximal convex chains.

Each such maximal convex chain of $\mathcal{P}'$ consists of at most three vertices that might all be convex in $\mathcal{P}'$. Observe though, that the last vertex v_j of a chain C is either a reflex vertex or also the first vertex of another maximal convex chain $C' = \langle v_j, \ldots, v_k \rangle$ of $\mathcal{P}'$. Hence, we need to count only at most two convex vertices per chain. Therefore, $\mathcal{P}'$ has at most $6r + 2$ convex vertices.

Finally, observe that we can apply the same strategy to a polygon $\mathcal{P}$ with holes $\mathcal{H}_i$ by considering each hole as a simple polygon itself. Recall that our approach simplifies a polygon, preserving geodesics inside the polygon. We independently pseudo-triangulate the interior of each hole $\mathcal{H}_i$ (in time linear in the number of vertices of $\mathcal{H}_i$) and apply the simplification strategy. For each $\mathcal{H}_i$ we obtain a polygon whose complexity is proportional to the number of convex vertices of (the polygon) $\mathcal{H}_i$ and preserves geodesics outside $\mathcal{H}_i$ (i.e., paths that consider $\mathcal{H}_i$ as an obstacle). Thus, the resulting simplification preserves geodesics inside $\mathcal{P}$. As each convex vertex of $\mathcal{H}_i$ is a reflex vertex of $\mathcal{P}$, the complexity of this simplification will be proportional to the number of vertices that are reflex in $\mathcal{P}$. □

Remark. Note that with this process we can explicitly give the coordinates of $\mathcal{P}'$. If the coordinates of the vertices of $\mathcal{P}$ are rational, then the coordinates used for $\mathcal{P}'$ are rational as well. Moreover, explicitly representing these vertices needs at most a constant times the number of bits used by input vertices. Alternatively, we can store the simplification in an implicit form: we store the first and last vertices of each simplified chain, allowing an easy identification between each connected component of $\mathcal{P}' \setminus \mathcal{P}$ and the simplified chains. In either case, at most $O(r)$ space is needed (for constant size vertex representation).

While our result is asymptotically optimal, we do not construct a subsuming polygon with the minimal number of vertices. Given a polygon $\mathcal{Q}$ with h holes, Guibas et al. [14, Theorem 5] show how to construct a simple polygon $\mathcal{P}$ with its boundary inside $\mathcal{Q}$ such that $\mathcal{P}$ contains all holes and has only $O(h)$ more vertices than the (probably non-simple) optimum. We note that, after knowing that the number of vertices of a minimal subsuming polygon is in $O(r)$, also the method by Guibas et al., after careful modification, could be used for constructing a subsuming polygon within the given bounds when applying it to each pocket.

However, our constructive proof of the bound gives a straight-forward algorithm that can be implemented using standard tools used for visibility problems in simple polygons.

4 Applications

Let $\mathcal{A}$ be an algorithm that receives a polygon $\mathcal{P}$ (and potentially other input I). We say that $\mathcal{A}$ is *subsuming* if the result of executing $\mathcal{A}$ with input $\mathcal{P}$ and I is equal to the result obtained when the input is $\mathcal{P}'$ and I, where $\mathcal{P}'$ is a polygon that subsumes $\mathcal{P}$. Note that, if $\mathcal{A}$ is subsuming, so will any other algorithm that solves the same problem (thus we say that the problem is *simplifiable*).

Theorem 2. *Let $\mathcal{P}$ be a polygon of size n with r reflex vertices and let I be additional input of size m. Let $\mathcal{A}(\mathcal{P}, I)$ be an algorithm that solves a simplifiable problem and runs in $T(n,m)$ time using $S(n,m)$ space. Then, $\mathcal{A}(\mathcal{P}, I)$ can be modified to run in $O(n + T(r,m))$ time and $O(n + S(r,m))$ space.*

In the following we present several applications for Theorem 2 which show the versatility of our approach. Most of these problems can be stated in terms of shortest paths, which immediately implies that the problems are simplifiable. For each task, we briefly state the problem, mention the (to the best of our knowledge) fastest existing algorithm, and explain how our approach helps to reduce the running time.

4.1 Shortest Paths

Computing the shortest path that avoids a series of obstacles and connects two given points belongs to the most fundamental problems in computational geometry. When looking for the shortest path between two given points inside a simple polygon $\mathcal{P}$ of size n, the currently best known algorithm is due to Guibas and Hershberger [12]: they provide a method that, after an $O(n)$ time preprocessing, can report the geodesic distance between any two points in $\mathcal{P}$ in $O(\log n)$ time. If the geodesic is to be reported, their method needs $O(k + \log n)$ time instead, where k is the number of vertices of the path. By applying our polygon simplification strategy, we reduce the query time to $O(\log r)$.

Corollary 1. *We can preprocess a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex in $O(n)$ time and space, such that for any two points $p, q \in \mathcal{P}$ we can determine their geodesic distance $|\pi_{\mathcal{P}}(p,q)|$ in $O(\log r)$ time. Moreover, the geodesic path can be reported in $O(k + \log r)$ time, where k is the number of vertices of $\pi_{\mathcal{P}}(p,q)$.*

Many variations of the above problem have been studied in the literature (see [21] for a survey). Among several results, we highlight the algorithm of Hershberger and Suri [16] for computing shortest paths in the case where holes are also present. The running time of their preprocessing algorithm is bounded by $O(n \log n)$, which can again be improved by our approach.

Corollary 2. *We can preprocess a polygon $\mathcal{P}$ with holes of n vertices out of which $r > 0$ are reflex in $O(n + r \log r)$ time and space, such that for any point $p \in \mathcal{P}$ we can determine the geodesic distance $|\pi_{\mathcal{P}}(p, q)|$ with respect to a fixed point $q \in \mathcal{P}$ in $O(\log r)$ time. Moreover, a geodesic path can be reported in $O(k + \log r)$ time, where k is the number of vertices of $\pi_{\mathcal{P}}(p, q)$.*

Shortest path computation has also been studied in other metrics, like, for example, the L_1 metric [20]. Although the proposed algorithm claims a running time of $O(n \log n)$, it is easy to see that convex vertices only contribute a linear fraction to the running time. Thus, the running time of the algorithm in [20] can be bounded by $O(n + r \log r)$ using a simple counting argument.

4.2 Geodesic Hull

A set $S \subseteq \mathcal{P}$ is called *geodesically* (or relative) convex if and only if for any $p, q \in S$, it holds that their geodesic $\pi_{\mathcal{P}}(p, q)$ is in S. The *geodesic hull* of a set $S \subseteq \mathcal{P}$ is defined as the (inclusion-wise) smallest geodesically convex set that contains S.

Given a set S of m points and a simple polygon $\mathcal{P}$ of n vertices, Toussaint [27] studied the problem of determining whether S is geodesically convex, and — if not — computing its geodesic hull. The proposed algorithm runs in $O((n + m) \log (n + m))$ time, which we can reduce with our approach.

Corollary 3. *Given a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex, we can compute the geodesic hull of a given set S of m points in the interior of $\mathcal{P}$ in $O(n + (m + r) \log(m + r))$ time using $O(n + m)$ space.*

4.3 Separating Point Sets

Given two sets R and B of m points in $\mathbb{R}^2$ and a geometric object ζ (usually a line) that partitions the plane into two components, we say that ζ *separates* R and B if each component of the plane only contains elements of one of the two sets. The extension of separability to polygons with holes was studied by Demaine et al. [9]. They showed that finding the minimum number of chords that separate the two sets inside a polygon with holes is NP-hard.

However, if $\mathcal{P}$ is a simple polygon, the problem becomes easier: in [9] the authors give necessary and sufficient conditions for the existence of a separating geodesic in simple polygons, which results in an algorithm that runs in $O((n + m) \log (n + m))$ time to determine the separability of R and B. The combination of their result with our simplification technique reduces the dependency in n.

Corollary 4. *Given a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex, and two sets R and B of m points each, we can determine whether or not R and B are separable by a geodesic (and find a separating geodesic, if any exists) in $O(n + (m + r) \log(m + r))$ time using $O(n + m)$ space.*

4.4 Triple Orientation

Many algorithms for point sets in the plane only use the orientations of point triples, i.e., a ternary predicate $p(a,b,c)$. In analogy to unconstrained point sets, the orientation of a point triple inside a simple polygon is defined via the order in which the points appear on the geodesic hull of the triple [2].

Corollary 5. *We can preprocess a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex in $O(n)$ time and space so that, for any three points $a, b, c \in \mathcal{P}$, we can determine their orientation in $O(\log r)$ time.*

In [2], it was shown that each point set in a simple polygon corresponds to an *abstract order type*, a generalization of point set order types in the plane (roughly speaking, an order type is an equivalence class of point sets w.r.t. the predicate p). Algorithms that operate on abstract order types can be applied in this setting, using $O(\log r)$ time per orientation test. For example, using the results of [3] one can compute a halving geodesic through a given point of a set S of m points, and also the geodesic hull edges stabbed by (the extension of) a geodesic through two given points in $O(n + m \log r)$ time.

4.5 Voronoi Diagram

Voronoi diagrams are another fundamental data structure in computational geometry, hence, it is no surprise that the geodesic variant was studied in the late 80s [5]. Since then the algorithms have been improved, and the currently best algorithm is due to Papadopoulou and Lee [23]. The furthest-site Voronoi diagram has also been studied in geodesic environments [6]. The fastest algorithms for computing either diagram run in $O((n+m)\log(n+m))$ time, and use $O(n+m)$ space [6,23]. It is easy to see that, for any point $q \in \mathcal{P}$, its nearest or furthest site (w.r.t. geodesic distance) will be the same in $\mathcal{P}$ and in any polygon that subsumes $\mathcal{P}$. Thus, in principle our approach can be used. Since the boundary is part of the Voronoi diagram, some post-processing will be necessary for this problem.

Corollary 6. *Given a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex, and a set S of m sites, we can compute the nearest- and furthest-site geodesic Voronoi diagram of S with respect to $\mathcal{P}$ in $O(n + (m+r)\log(m+r))$ time using $O(n+m)$ space.*

4.6 Geodesic Diameter, Median, and Maximum Distance

Given a set S of n sites in a simple polygon $\mathcal{P}$, the *geodesic median* is the site of S that minimizes the maximum geodesic distance to points of S. The *geodesic diameter* of S is the maximal distance between any two sites of S. If we are given two sets of sites S_1 and S_2 instead, their *maximum geodesic distance* is defined as the largest distance between points of different sets. Toussaint [27] introduced these concepts and gave algorithms on how to compute them. The running times of the proposed algorithms range between $O((n+m)\log(n+m))$ and $O(n^2+m^2)$. As pointed out by Toussaint, these problems can also be solved by computing the furthest-site geodesic Voronoi diagram and making $O(m)$ queries. If we simplify the polygon before making the queries, we obtain the following result.

Corollary 7. *Given a simple polygon $\mathcal{P}$ of n vertices out of which $r > 0$ are reflex, and two sets S_1, S_2 of m sites, we can compute the geodesic median, the geodesic diameter of S_i (for $i \leq 2$), and the maximum geodesic distance between S_1 and S_2 in $O(n + (m + r)\log(m + r))$ time using $O(n + m)$ space.*

5 Conclusion

While asymptotically tight, our method does not construct optimal subsuming polygons, i.e., with the minimum number of vertices. An optimal subsuming polygon consists of several disjoint paths with a minimum overall number of vertices. In general, the minimum link path problem is 3SUM-hard [22] even for a single path. However, the reduction used in [22] is not applicable to our restricted setting where the paths are in a domain without holes. Inside simple polygons, the minimum link path problem is solvable in linear time [26] for a single path.

Guibas et al. [14] show that finding a minimum link simple polygon having its boundary inside a given region R is NP-hard, but they point out that their reduction requires holes in R on both sides of the polygon boundary. The pairwise-disjoint link paths problem inside a simple polygon is discussed by Gupta and Wenger [15]. They give a constant-factor approximation algorithm, and ask whether there exists a polynomial-time algorithm for computing the optimum. We state the same question for our restricted setting: Given a simple polygon $\mathcal{P}$, what is the complexity of constructing a subsuming polygon with the least number of (convex) vertices?

References

1. Aichholzer, O., Hackl, T., Korman, M., Pilz, A., Vogtenhuber, B.: Geodesic-preserving polygon simplification. ArXiv e-prints (2013), arXiv:1309.3858
2. Aichholzer, O., Korman, M., Pilz, A., Vogtenhuber, B.: Geodesic order types. Algorithmica (to appear, 2013)
3. Aichholzer, O., Miltzow, T., Pilz, A.: Extreme point and halving edge search in abstract order types. Computational Geometry: Theory and Applications 46(8), 970–978 (2013)
4. Arkin, E.M., Chiang, Y.-J., Held, M., Mitchell, J.S.B., Sacristan, V., Skiena, S., Yang, T.-H.: On minimum-area hulls. Algorithmica 21(1), 119–136 (1998)
5. Aronov, B.: On the geodesic Voronoi diagram of point sites in a simple polygon. In: SoCG, pp. 39–49 (1987)
6. Aronov, B., Fortune, S., Wilfong, G.T.: The furthest-site geodesic Voronoi diagram. Discrete and Computational Geometry 9, 217–255 (1993)

7. Bern, M.W., Eppstein, D.: Mesh generation and optimal triangulation. In: Computing in Euclidean Geometry. Lecture Notes Series on Computing, vol. 4, pp. 47–123. World Scientific (1995)
8. Bose, P., Demaine, E.D., Hurtado, F., Iacono, J., Langerman, S., Morin, P.: Geodesic ham-sandwich cuts. In: SoCG, pp. 1–9 (2004)
9. Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Meijer, H., Overmars, M.H., Whitesides, S.: Separating point sets in polygonal environments. International Journal of Computational Geometry and Applications 15(4), 403–420 (2005)
10. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2), 112–122 (1973)
11. Ghosh, S.: Visibility Algorithms in the Plane. Cambridge University Press, New York (2007)
12. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. Journal of Computer and System Sciences 39(2), 126–152 (1989)
13. Guibas, L.J., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2, 209–233 (1987)
14. Guibas, L.J., Hershberger, J., Mitchell, J.S.B., Snoeyink, J.: Approximating polygons and subdivisions with minimum link paths. Journal of Computer and System Sciences 3(4), 383–415 (1993)
15. Gupta, H., Wenger, R.: Constructing pairwise disjoint paths with few links. ACM Transactions on Algorithms 3(3) (2007)
16. Hershberger, J., Suri, S.: An optimal algorithm for euclidean shortest paths in the plane. SIAM Journal of Computing 28(6), 2215–2256 (1999)
17. Hershberger, J., Snoeyink, J.: Computing minimum length paths of a given homotopy class. Computational Geometry: Theory and Applications 4, 63–97 (1994)
18. Hertel, S., Mehlhorn, K.: Fast triangulation of simple polygons. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 207–218. Springer, Heidelberg (1983)
19. Melkman, A.A.: On-line construction of the convex hull of a simple polyline. Information Processing Letters 25(1), 11–12 (1987)
20. Mitchell, J.S.B.: L_1 shortest paths among polygonal obstacles in the plane. Algorithmica 8, 55–88 (1992)
21. Mitchell, J.S.B.: Shortest paths and networks. In: Handbook of Discrete and Computational Geometry, 2nd edn., pp. 607–642. Chapman & Hall/CRC (2004)
22. Mitchell, J.S.B., Polishchuk, V., Sysikaski, M.: Minimum-link paths revisited. ArXiv e-prints (2013), arXiv:1302.3091
23. Papadopoulou, E., Lee, D.T.: A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains. Algorithmica 20(4), 319–352 (1998)
24. Rote, G., Santos, F., Streinu, I.: Pseudo-triangulations—a survey. In: Surveys on Discrete and Computational Geometry—Twenty Years Later. Contemporary Mathematics, pp. 343–410 (2008)
25. Speckmann, B., Tóth, C.D.: Allocating vertex π-guards in simple polygons via pseudo-triangulations. Discrete and Computational Geometry 33(2), 345–364 (2005)
26. Suri, S.: A linear time algorithm for minimum link paths inside a simple polygon. Computer Vision, Graphics, and Image Processing 35(1), 99–110 (1986)
27. Toussaint, G.T.: Computing geodesic properties inside a simple polygon. Revue D'Intelligence Artificielle 3(2), 9–42 (1989)

Space-Efficient and Data-Sensitive Polygon Reconstruction Algorithms from Visibility Angle Information

Jinhee Chun, Ricardo Garcia de Gonzalo, and Takeshi Tokuyama

GSIS Tohoku University
{jinhee,tokuyama}@dais.is.tohoku.ac.jp

Abstract. We propose a linear working space algorithm for reconstructing a simple polygon from the visibility angle information at vertices up to similarity. We also modify the algorithm such that its running time is sensitive to the size of visibility graph and also the diameter of a triangulation of the polygon.

1 Introduction

We consider a variation of the polygon reconstruction problem. The difficulty of the polygon reconstruction problem depends on input information, and it often becomes NP-hard ([2,3]). We particularly are interested in finding the shape of a simple polygon from angle information.

If the polygon is a triangle, we learn in elementary school that the shape and scale of a triangle is determined by either (1) the lengths of three edges (2) two angles and a length of an edge, or (3) lengths of two edges and the angle between those edges. In high school, we learn that the cosine theorem and sine theorem give explicit way to compute other information from one of those minimal sets of information. If we ingore scale, two angles are enough to determine the shape.

For a quadrangle, its shape cannot be determined from the set of four angles between adjacent edges. For example, a square and any rectangle have exactly the same set of four angles (i.e., $\pi/2$). However, if each vertex can see all other three vertices and the two visibility angles are measure to be $(\pi/4, \pi/4)$, we can confidently say that the shape is a square.

It is a natural question to generalize the above observation. Given a simple polygon P, a pair (u, v) is called a visiblity edge if the line segment uv lies in the closure of P. We often call the ray from u towards v a visibility ray if (u, v) is a visibility edge, and say v is visible from u and vice versa (Fig. 1). Note that v is visible from u in the right picture although uv touches the boundary at t. The visibility graph of P is a graph $G(P) = (V(P), E(P))$, where $V(P)$ is the set of vertices of P, and $E(P)$ is the set of visibility edge.

The visibility angle information is the list of angles around each vertex in the visibility graph drawn by line segments as shown in the left picture of Fig.2. We say two polygons have the same shape if one can be transformed to the other by applying a similarity translation. We would like to ask how to retrieve the

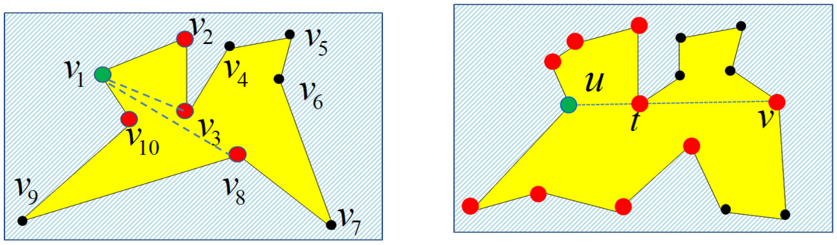


Fig. 1. Visibility rays from v_1 (left). Red vertices are visible from u (right).

shape of a simple polygon bounding a polygonal region P from the visibility angle information.

Let us give the explicit formulation of the problem. We consider a simple polygonal region P. The circular list $\{v_1, v_2, \dots, v_n\}$ of vertices of the boundary polygon ∂P is given in the clockwise ordering such that v_i is adjacent to v_{i-1} and v_{i+1} (index is considered modulo n so that $v_0 = v_n$). Each vertex v_i has a list of visibility angles $\xi(v_i, 1), \xi(v_i, 2), \dots \xi(v_i, d(v_i) - 1)$ where $\xi(v_i, j)$ is the angle between rays towards the j-th and $(j+1)$-th visible vertices in the clockwise ordering, and $d(v_i)$ is the number of visible vertices from v_i. We sometimes denote $\xi(v, -j)$ for $\xi(v, d(v) - j)$ for convenience sake. By definition, v_{i+1} and v_{i-1} are the first and last visible vertices from v_i in the clockwise order. We assume in this paper that the outer angle of P at each v_i is also given as a part of input; in other words, we have information of the partition of the circular visibility about each node. Our task is to reconstruct the shape of the polygon P from this visibility angle information.

To get intuition of the problem, imagine the case where the points v_i are visual sensors glittering and located at all vertices of a polygonal room, and they can see other indistinguishable sensors if the right rays go straightly in the room. Sensors can measure the visual angles between them, but cannot measure the distances. Our task is to find the shape of the room.

Disser *et al.* [5] first considered this problem, and showed that the shape of the polygon is uniquely determined. They gave an $O(n^3 \log n)$ time algorithm to compute it. The time complexity has been improved to $O(n^2)$ by Chen and Wang [4]. Both of the algorithms are based on clever dynamic programming to reconstruct the visibility graph of P.

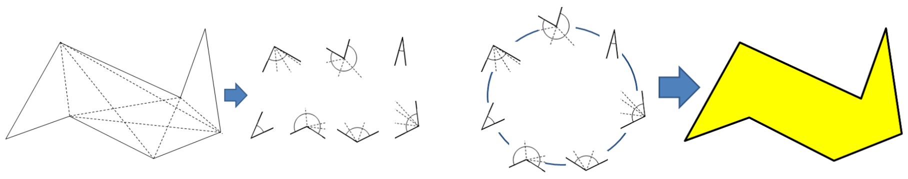


Fig. 2. Visibility angle information corresponding to the visibility graph of a polygon (left), and polygon reconstruction from the visibilty angle information (right)

One drawback of those algorithms is the high space complexity. This is becase they reconstruct the visibility graph. The visibility graph needs $\Omega(n^2)$ space (e.g. if P is a convex polygon) to store, and the dynamic programing algorithms need quadratic space even if the visibility graph has a small size. Indeed, there is no way to improve $O(n^2)$ time and space complexity given by Chen and Wang [4]l if we explicitly construct the visibility graph. However, it seems to be overkill to compute the visibility graph in order to find the shape of the polygon.

In this paper, we give a different idea to reconstruct the polygon. Instead of using a visibility graph, we find a triangulation of P. We first give an algorithm that runs in $O(n^2 \log n)$ time and $O(n)$ working space in the worst case. Note that the $\Omega(n^2)$ space is necessary to store the input information, since the input size m in the word RAM model is asymptotically the number of visible pairs of vertices, and there are $n(n-1)/2$ visible pairs if the polygon is a convex polygon. However, the input data can be given in a read-only memory with fast access, and we are interested in reducing working memory.

Next, we improve the algorithm if the input size is smaller than n^2. The input size m of the problem is linear in the number of edges of the visibility graph. Although m can become quadratic in n, it was a research issue in the literature to compute the visibility graph of a polygon in time sensitive to the size of the visibility graph, and Hershberger [8] solved it by giving an optimal $O(m)$ time algorithm. Therefore, it is also valuable to give an algorithm sensitive to m for our reconstruction problem. We modify the algorithm to reduce the time complexity to $O(n \log^4 n + m \log^2 n)$, which is faster than $O(n^2)$ if $m = o(n^2/\log^2 n)$.

Then, we investigate the dependency of the running time of algorithms on geometric properties of the polygon. One surprising fact is that our algorithms often find the polygon without reading the whole input information, and the running time can become $o(m)$ for several types of input polygons. Indeed, the running time of the first algorithm is $O(nh \log n + n \log^2 n)$ if the obtained triangulation has depth h (i.e., its dual graph is a tree of height h). In particular, $h = O(n_0 + \log n)$ if P happens to be a polygon with n_0 concave vertices. Note that the algorithm terminates when a unique candidate simple polygon consistent with the input data read so far is identified, and we exclude the complexity (it can be done in $O(m \log n)$ time with $O(n)$ space) to certify that the whole input data is really consistent with the output polygon.

2 A Linear Space Algorithm

2.1 Remarks on Geometric Settings for Solving the Problem

Recall that a visibility edge uv may be tangent to the boundary polygon ∂P from its inside. If uv touches at t on ∂P as shown in the right picture in Figure 1, u can both see t and v, and it has a 0 angle $\angle tuv$ in its list of visibility angles. We assume this convention throughout this paper.

We remark that our algorithm may fail in a different setting where we do not allow u and v visible to each other if there is another vertex or an edge touching uv. However, previous known algorithms do not work in that setting, either.

We allow P to be degenerate in the sense that ∂P may be tangent to itself. Although the input polygon is a simple polygon, we use this convention since it may happen that we need to handle such a subpolygon of P in our algorithm. We often refer to a polygon for both the polygonal region and the polygonal cycle.

Although nodes do not share a universal direction, we can provide it to them with $O(n)$ time computation from the counterclockwise ordering of vertices and outer angles. In other words, we can find all slopes of edges of the polygon if we assume that v_1v_2 is horizontal, since we can find the slope of $v_{i-1}v_i$ by summing the outer angles up to the vertex v_i.

We note that if we could assume that there is no pair of visibility edges parallel to each other, then the problem could be easily solved without using serious geometry by sorting (or hashing) the slopes of rays and find the unique pairing of them. In other words, existence of parallel visibility edges is essential to the difficulty of the problem.

2.2 Idea of the Algorithm

It is well-known that a simple polygon has a triangulation, and the dual graph of a triangulation of is a tree of maximum degree 3. A triangle corresponding to a leaf of the tree is called an *ear* of the triangulation. If we remove an ear $\triangle$ from the polygon, we obtain a smaller simple (possibly degenerate) polygon $P \setminus \triangle$. This operation is called *ear clipping*. We can use ear clipping iteratively to triangulate a polygon as illustrated in Fig. 3. If we know the positions of vertices of the polygon P explicitly, the ear clipping triangulation algorithm works in $O(n^2)$ time [6].

Our main goal is to simulate this algorithm using only the visibility angle information. Initially, we test triangle $\triangle v_{i-1}v_iv_{i+1}$ for each $i = 1, 2, \ldots, n$ whether it is an ear or not. Indeed, $\triangle v_{i-1}v_iv_{i+1}$ is an ear if and only if it does not contain another vertex in its interior and the inner angle $\angle v_{i-1}v_iv_{i+1} \leq \pi$. We note that the equality can happen since we allow degeneracy. Since the second condition is easy to verify, we consider the first condition.

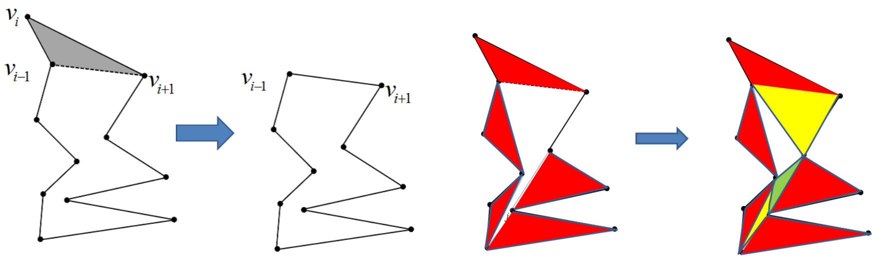


Fig. 3. Ear Clipping (left) and ear clipping triangulation(right)

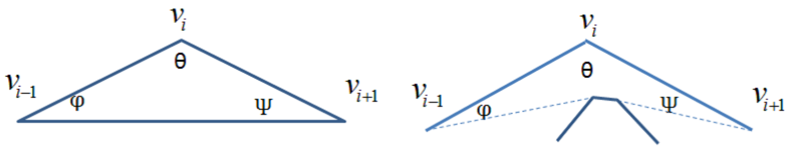


Fig. 4. Triangle-certificate equality is sufficent and necessary

Let $\xi(v,k)$ (resp. $\xi(v,-k)$) be the k-th angles around v in CW (resp. CCW) ordering starting from the ray to its right (resp. left) adjacent vertex in P. Here, CW and CCW means clockwise and counter-clockwise, respectively.

In particular, if $k = 1$, they are the first angles in CW and CCW ordering, respectively. We set $\theta = \angle v_{i-1}v_iv_{i+1}$, $\varphi = \xi(v_{i-1},1)$ and $\psi = \xi(v_{i+1},-1)$ as shown in Fig. 4, and have the following elementary lemma:

Lemma 1. *The triangle $\triangle v_{i-1}v_iv_{i+1}$ is an ear if and only if $\varphi+\theta+\psi=\pi$. We call this equality the* triangle-certificate equality*.*

Once we find an ear, we remove it and update the boundary chain of P, and process it.

Suppose that we have a boundary chain $v_{q(1)}, v_{q(2)}, \ldots v_{q(k)}$ of a polygonal region Q during the processing of our algorithm, where $q(1), q(2) \ldots q(k)$ is a subsequence of $1, 2, \ldots n$. Let $e(i,Q)$ be the edge of Q between $v_{q(i)}$ and $v_{q(i+1)}$, and let $P(i)$ be the polygonal chain of P cut off from Q by the edge $e(i,Q)$. Since our algorithm removes ears one by one, $P(i)$ has been triangulated. Hence the locations of vertices of $P(i)$ are known relative to $e(i,Q)$, i.e. we know the exact location once we fix the position and scale of $e(i,Q)$. We also have all slopes of $e(i,Q)$ at this stage assuming that the initial edge v_1v_2 is horizontal.

Thus, one may consider that we can process Q similar to the original polygon P. Unfortunately, the annoying problem is that the angle information at each vertex has not been updated at removal of an ear, and the rays (and corresponding angles) towards removed vertices still remain. Thus, the naive iteration of the process will fail.

2.3 Ear Detection Routine

We describe how to detect an ear in our algorithm when some ears have been already clipped. See the left picture of Fig.5.

Each vertex $v_{q(i)}$ identifies the rays towards its neighbor vertices $v_{q(i-1)}$ and $v_{q(i+1)}$, since they can be recorded when the ears were clipped at the edges $e(i,Q)$ and $e(i-1,Q)$. Thus, we can easily compute $\theta = \angle v_{q(i-1)}v_{q(i)}v_{q(i+1)}$. Suppose that we want to decide whether the triangle $\triangle v_{q(i-1)}v_{q(i)}v_{q(i+1)}$ is an ear of Q. We slightly abuse the notation, and let $\xi(v_{q(i-1)},k)$ be the k-th visibility angle around $v_{q(i-1)}$ starting from the edge $v_{q(i-1)}v_{q(i)}$. Consider the ray r_φ emanated from v_{q-1} defining the angle $\varphi = \xi(v_{q(i-1)},1)$. There are three cases:

Case 1. The ray r_φ goes to $v_{q(i+1)}$. Consequently, the triangle is an ear.

Case 2. It goes to a vertex in $P(i)$ excluding $v_{q(i+1)}$.

Case 3. It goes to other vertex. The triangle cannot be an ear for this case.

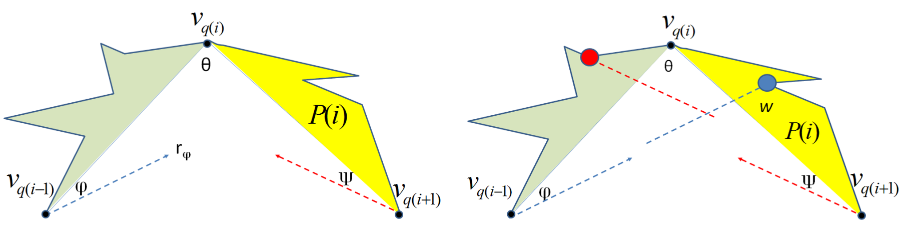


Fig. 5. The destinations of rays emanated from $v_{q(i-1)}$ and $v_{q(i+1)}$

We claim that we can decide which of the three cases happens. We write r for r_φ from now on. Since know the direction of the ray r, we can compute the projection $Z = \text{Proj}(P(i), r)$ of $P(i)$ with respect to the ray r, which shows the visibility of the polygonal chain $P(i)$ to the direction of r. The vertex defining the ray r should have its projection in Z, and it should be the next (projected) vertex w to $v_{q(i)}$ in the projection as shown in the right picture of Fig. 5. Note that case 1 can happen only if $w = v_{q(i+1)}$.

Then, w should have a ray to the opposite direction to r (however, we need not examine the angle data of w) if nothing blocks the ray from $v_{q(i)}$ towards w.

If Case 1 or Case 2 happens, r must go through w, and we can determine the position of $v_{q(i-1)}$ explicitly as its intersection with the ray from $v_{q(i)}$ towards $v_{q(i-1)}$ as shown in Fig. 6. Since the location of w is known (relative to $e(i, Q)$), we now know the exact position of $v_{q(i-1)}$ relative to $e(i, Q)$.

Therefore, we have an estimation α of ratio of lengths of edges $e(Q, i-1) = v_{q(i-1)}v_{q(i)}$ to $e(Q, i) = v_{q(i)}v_{q(i+1)}$. We do the same operation from $v_{q(i+1)}$ to have another estimation α' of the same ratio.

If $\alpha \neq \alpha'$ as shown in the left picture of Fig. 6, we detect that the triangle cannot be an ear, and hence we stop processing this triangle and move to the next one. Therefore, let us assume that $\alpha = \alpha'$ as shown in the right picture of Fig. 6.

Lemma 2. *If $\alpha = \alpha'$, α is the actual ratio of lengths of edge $v_{q(i-1)}v_{q(i)}$ to $v_{q(i)}v_{q(i+1)}$ even if the triangle is not an ear. As a consequence, the shape of the triangle $\triangle v_{q(i-1)}v_{q(i)}v_{q(i+1)}$ is explicitly determined.*

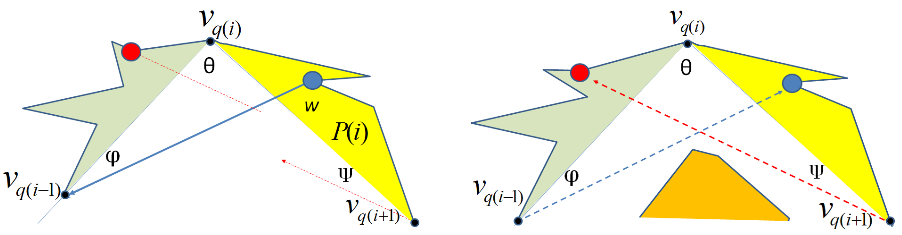


Fig. 6. Finding the position of $v_{q(i-1)}$

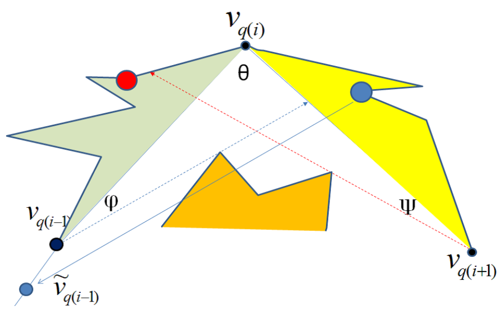


Fig. 7. Illustration of proof of Lemma 2

Proof. Suppose that the ray r goes to a vertex x inside the triangle as shown in Fig.7, where Case 3 happens. The ray should be tangent to ∂P at x, since the ray defines the smallest angle and ∂P cannot cross the segment $v_{q(i-1)}v_{q(i)}$.

Assume that the real ratio α_0 of $v_{q(i-1)}v_{q(i)}$ to $v_{q(i)}v_{q(i+1)}$ is larger than α. Then, the ray from $v_{q(i-1)}$ to w has smaller angle than φ. If the ray is blocked by some edge, one of its endpoint must define another smaller angle at $v_{q(i-1)}$, which is a contradiction. Thus, $\alpha \geq \alpha_0$ as shown in the figure (i.e., the estimated position $\tilde{v}_{q(i-1)}$ is farther than the real position $v_{q(i-1)}$). If we consider the same argument at $v_{q(i+1)}$, we have $1/\alpha' \geq 1/\alpha_0$. Thus, if $\alpha = \alpha'$, $\alpha = \alpha_0$, and we correctly find the shape. □

Now, we can explicitly compute the visible vertices of $P(i)$ from $v_{q(i-1)}$, and know which rays point to vertices in $P(i)$. Then, we compare it with the visibility angles around $v_{q(i-1)}$. The rays that are not blocked should give a prefix sequence of angles around $v_{q(i-1)}$ in the CW ordering. Thus, we check the rays one by one whether it goes to a vertex of $P(i)$. Now we know the last ray from $v_{q(i-1)}$ pointing to a vertex of $P(i)$. It points to $v_{q(i+1)}$ if and only if the triangle is an ear.

Note that it may happen that a ray from $v_{q(i-1)}$ towards a vertices y in $P(i)$ also reaches a point x of Q. We consider the first such ray in the list of visibility rays at $v_{q(i-1)}$, and assume all rays prior to it go to vertices of $P(i)$. Then, the ray must be tangent to Q at x. Because of our assumption for the visibility of degenerate case, $v_{q(i-1)}$ has an angle 0 in the angle list there (or, equivalently it has multiple rays to this direction, and we know the multiplicity by our assumption about degeneracy), and hence we can detect that the current ray is the last ray reaching to a vertex of $P(i)$.

Since the computation of projection and visibility from a point can be done in $O(|P(i)| \log P(i))$ time (we omit its details since it is forklore), the algorithm finds at least one new ear in $O(n \log n)$ time. Thus, the total time complexity is $O(n^2 \log n)$ to find $n-3$ ears to compute the triangulation. The algorithm clearly uses only inear working space (we omit details). Thus, we have the following theorem:

Theorem 1. *The reconstruction of a simple polygon from visibility angle date can be done in linear space and $O(n^2 \log n)$ time.*

3 Algorithm Sensitive to the Size of Visibility Graph

Although the time complexity of the problem has been analyzed as functions of n in the previous section, the input size of the problem is in proportional to the number of visibility angles, which we denote by m. Therefore, it is desirable to analyze the complexity by using n and m.

3.1 Stage-Wise Greed Ear-Clipping Triangulation Algorithm

The algorithm given in the previous section finds an ear after testing $O(n)$ triangles. Therefore, it needs to test $O(n^2)$ triangles in the worst case. This can be avoided by modifying the strategy to clip ears.

We consider a variant of the ear-clipping algorithm, in which only $O(n)$ triangles are examined whether they are ear or not. We divide the algorithm into stages. In each stage, we check triangles one by one in the clockwise ordering. In the first round, we check each triangle one by one whether it is an ear or not, but if we find an ear $\triangle v_{i-1}v_iv_{i+1}$, we skip its adjacent triangle (since the edge v_iv_{i+1} has been removed) and next check $v_{i+1}v_{i+2}v_{i+3}$.

In the j-th round for $j \geq 2$, we only check triangles adjacent to at least one newly created edge corresponding to the ears removed in $(j-1)$-th round (See Fig. 3 to get intuition, where colors indicate stages). We use the same skipping rule once we find a new ear. The following lemma justifies the strategy of the algorithm.

Lemma 3. *In the j-th round, no triangle that does not use an edge created in the $(j-1)$-th round can be an ear.*

Proof. Suppose that a triangle $\triangle$ formed by consecutive triple of vertices does not use an edge created in the $(j-1)$-th round. It was checked once in a stage just after one of its edges was created (or in the first stage). Since it was not an ear at that time, it intersected the complement of the polygonal region. However, the complement of the polygonal region only grows in our ear clipping procedure. Thus, the triangle cannot become an ear forever. $\square$

If we find m_i ears in the i-th round, then we only need to check $2m_i$ triangles in the $(i+1)$-th round. Thus, the number of examined triangles is at most twice of the number of removed ears. Since we remove $n-1$ ears in total, the number of triangles we examine in the algorithm is $O(n)$.

3.2 Reducing the Time Complexity by Using Ray-shooting Query

We next reduce the work at each triangle by using *ray shooting query*. We preprocess a polygonal chain (in general, a set of segments) so that given any query ray, the ray shooting query answers the first intersecting segment with the ray (Fig.8, left picture). The following theorem is known.

Theorem 2 (Goodrich *et al.*[7]). *Given a set of n segments forming edges of a connected planar graph drawing, we can construct a data structure to perform ray shooting query in $O(\log^2 n)$ time, and it can be dynamically updated in $O(\log^2 n)$ time per insertion and deletion.*

Now we describe the algorithm. The algorithm is almost same as the one presented in the previous section. The difference is that we use ray shooting query to find the endpoint of a ray from $v_{q(i-1)}$ towards $P(i)$.

The first task is to find the position of $v_{(q-1)}$. For the purpose, we need to find the partner vertex w of the ray r defining $\varphi = \xi(v_{q(i-1)}, 1)$ in $P(i)$. In order to simplify the argument, we assume that r is a vertical ray coming from above, and the visible part of $P(i)$ is projected to the horizontal interval $I = (s,t) = (\mathrm{Proj}(v_{q(i)}), \mathrm{Proj}(v_{q(i+1)}))$ on the x-axis, where Proj means the vertical projection. Also, we assume $\xi(v_{q(i-1)}, 1) > 0$, since we can handle the special case where the angle is zero independently (we omit its details).

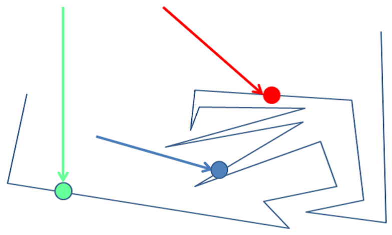
In this setting, the vertex w must be the leftmost visible vertex of $P(i)$ except $v_{q(i)}$ as illustrated in the right picture of Figure 8. Let $r(x)$ be the vertical ray parallel to r with the abscissa x. For an infinitesimally small positive real value ϵ, we consider the edge f_{s+} found by the ray shooting by $r(s+\epsilon)$. It is the unique edge of the chain $P(i)$ adjacent to $v_{q(i)}$ if its projection is in I; otherwise it is the edge visible from $v_{q(i)}$ to the direction of r, which we can compute by using the ray shooting query. Thus, we can find f_{s+} in $O(\log^2 n)$ time. Let x_w be the abscissa of w. The following lemma is easy to observe (see Fig. 8).

Lemma 4. *For any value $x \in I$, the ray shooting by $r(x)$ finds a point on f_{s+} if and only if $x_w > x$. Moreover, if the ray shooting finds a vertex u of $P(i)$, $u = w$ if and only if either $v_{q(i))}u$ is f_{s+} or the ray shooting from u finds a point on f_{s+}.*

By applying the above lemma, we can design a binary searching algorithm to find x_w if we have a sorted list of abscissa of vertices of $P(i)$.

Unfortunately, it takes $O(n_i \log n_i)$ time to obtain such a sorted list if $P(i)$ has n_i vertiices, and it is too expensive. Instead, we apply the *parametric searching* paradigm [9]. Using the general theory of the paradigm, if we can query in $O(T)$ time at each position of the ray, and can decide which side of the ray the vertex w lies, we can find the position of w in $O(T^2)$ time. Therefore, it takes $O(T^2) = O(\log^4 n)$ time to find the position w. Since we examine $O(n)$ triangles during our triangulation algorithm, this process needs $O(n \log^4 n)$ time in total.

Once we have w, we explicitly know the position of $v_{q(i-1)}$ relative to the edge $e(i,Q)$. Then, we perform the ray shooting from $v_{q(i-1)}$ to the directions of rays emanated from it one by one, until either we find $v_{q(i+1)}$ or we find a ray that



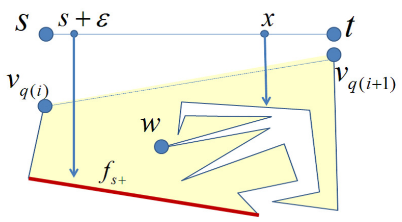


Fig. 8. Ray-shooting(left) and binary searching by ray shooting(right)

does not reach a vertex of $P(i)$. In the first case, we declare that we have found an ear, and in the second case, we declare that the current triangle is not an ear. We store a *finger* to indicate the position of the last ray to reach a vertex of $P(i)$, so that we can restart from the position of the finger if we need to do ray shooting from $v_{q(i-1)}$ later. Since the finger position does not go back, this process takes $O(m)$ ray shooting queries, and hence $O(m \log^2 n)$ time in total. Thus, the total time complexity spending ray shooting is $O(n \log^4 n + m \log^2 n)$.

Once we find an ear, two polygonal chains are merged, and the data structure of the ray shooting should be updated, where we insert edges in the smaller chain to the structure of larger one. Then, the total time complexity for update of this data structure for the whole process of our triangulation is $O(n \log^3 n)$ since each edge is inserted at most $\log n$ times. Thus, we have the following:

Theorem 3. *The polygon reconstruction problem can be solved in $O(n \log^4 n + m \log^2 n)$ time using $O(n)$ space.*

Our algorithms may terminate before reading whole data. We can check the consisteny of the rest of input data by applying ray shooting in the output polygon P for each ray. It can be done in $O(m \log n)$ time and $O(n)$ space if we apply a static ray shooting data structure [1].

4 Analysis Dependent on Properties of Polygons

The time complexities of our algorithms are $O(n^2 \log n)$ time and $O(n \log^4 n + m \log^2 n)$, respectively. However, the algorithm often runs much faster.

To get intuition, suppose that the polygon P is a convex polygon. where $m = n(n-1)/2$. If we apply the stage-wise clipping strategy in first algorithm, it reduces the number N of vertices at each stage to the half by finding $\lceil N/2 \rceil$ ears, and spend $O(n \log n)$ time for each stage. Thus, we have $O(\log n)$ stages, and the total time complexity is $O(n \log^2 n)$. We remark that it is important that the outer angles are given as a part of input for this analysis.

This can be generalized as follows: Consider the triangulation of the polygon P created by the stage-wise ear clipping algorithm. Its dual graph is a tree T, and let h be the height of the tree selecting any leaf node as its root. Then, we have the following.

Theorem 4. *We can compute P from the visibility angle information in $O(nh \log n)$ time.*

Proof. Let us consider the number of stages in the stage-wise ear-clipping algorithm. An important fact is that each ear clipped in a stage was connected to an ear clipped in the previous stage. Thus, if we trace back the clipping algorithm from the end to the begining, it starts with a single triangle $\triangle$, adding at least two ears to grow the polygon, and in each stage new ears are attached to ears created in the last stage. Thus, the number of stages is bounded by the length of path from $\triangle$ to a leaf of the tree. Since the length of such path is at least twice of the height of the tree, the number of stages is at most $2h$. Since the algorithm needs $O(n \log n)$ time for each stage, we have the theorem. $\square$

Corollary 1. *Suppose that the polygon has n_0 concave vertices and $n - n_0$ convex vertices. Then, we can compute P in $O(n \log^2 n + n_0 n \log n)$ time.*

Proof. The number n_0 is counted from the outer angle information, and the number of concave vertices never increases during the algorithm, since each inner angle is only decreased by clipping an ear. Let $n_1 = n - n_0$. Each of the n_1 convex vertices defines a triangle with its adjacent vertices. This triangle is an ear if there is no concave vertex inside it. Each concave vertex locates in at most two such triangles. Thus, among those n_1 triangles, at least $n_1 - 2n_0$ are ears. Thus, the algorithm clips at least $(n_1 - 2n_0)/2$ ears in one stage. While $n_1 \geq 4n_0$ holds, we have $(n_1 - 2n_0)/2 \geq n/5$, and the number of vertices is reduced to $4n/5$. Thus, after $O(\log n)$ stages, we have $n_1 < 4n_0$ and hence $n < 5n_0$. Thus, there are at most n_0 stages after that, and we have the complexity. □

5 Concluding Remarks

Since the expected height of a random treap is $O(\log n)$, average time complexity of our algorithm is $O(nh \log n) = O(n \log^2 n)$ if the trees generated by our algorithm obey the distribution of random treap.

We do not have an input construction to force the algorithm to spend quadratic running time even if $m = \Omega(n^2)$, and we suspect that the time complexity of the algorithm may be subquadratic in n for any polygon.

Acknowledgment. This work was supported by JSPS KAKENHI Grant Number 22300001, 24650001 and 25330002.

References

1. Agarwal, P.: Range Searching. In: Goodman, J., O'Rourke, J. (eds.) Handbook of Discrete and Comput. Geom., ch. 36, pp. 809–838. Chapman&Hall (2004)
2. Asano, T., Ghosh, S.K., Shermer, T.: Visibility in the Plane. In: Sack, J., Urrutia, J. (eds.) Handbook of Computational Geometry, ch. 19, pp. 829–876. Elseiver (2000)
3. Biedl, T., Durocher, S., Skneyink, J.: Reconstructing Polygons from Scanner Data. Theoretical Computer Science 412, 4161–4272 (2011)
4. Chen, D.Z., Wang, H.: An Improved Algorithm for Reconstructing a Simple Polygon from Its Visibility Angles. CGTA 45, 254–257 (2012)
5. Disser, Y., Mihalak, M., Widmayer, P.: A Polygon is Determined by Its Angles. CGTA 44, 418–426 (2011)
6. ElGindy, H., Everett, H., Toussaint, G.: Slicing an Ear Using Prune-and-Search. Pattern Recognition Letters 14(9), 719–722 (1993)
7. Goodrich, M.T., Tamassia, R.: Dynamic Ray Shooting and Shortest Paths in Planar Subdivisions via Balanced Geodesic Triangulations. J. Algorithms 23(1), 51–73 (1987)
8. Hershberger, J.: An Optimal Visibility Graph Algorithm for Triangulated Simple Polygons. Algorithmica 4, 141–155 (1989)
9. Salowe, J.S.: Parametric Searching. In: Goodman, J., O'Rourke, J. (eds.) Handbook of Discrete and Comput. Geom., ch. 43, pp. 969–982. Chapman&Hall (2004)

On the Edge Crossing Properties of Euclidean Minimum Weight Laman Graphs

Sergey Bereg[1], Seok-Hee Hong[2,⋆], Naoki Katoh[3,⋆⋆], Sheung-Hung Poon[4,⋆⋆⋆], and Shin-ichi Tanigawa[5,†]

[1] Department of Computer Science, University of Texas at Dallas, USA
besp@utdallas.edu

[2] School of Information Technologies, University of Sydney, Australia
shhong@it.usyd.edu.au

[3] Department of Architecture and Architectural Engineering, Kyoto University, Japan naoki@archi.kyoto-u.ac.jp

[4] Department of Computer Science & Institute of Information Systems and Applications, National Tsing Hua University, Taiwan, R.O.C
spoon@cs.nthu.edu.tw

[5] Research Institute for Mathematical Sciences, Kyoto University, Japan
tanigawa@kurims.kyoto-u.ac.jp

Abstract. This paper is concerned with the crossing number of Euclidean minimum-weight Laman graphs in the plane. We first investigate the relation between the Euclidean minimum-weight Laman graph and proximity graphs, and then we show that the Euclidean minimum-weight Laman graph is quasi-planar and 6-planar. Thus the crossing number of the Euclidean minimum-weight Laman graph is linear in the number of points.

1 Introduction

A graph G is called *Laman* if $|E(G)| = 2|V(G)| - 3$ and $|E(H)| \leq 2|V(H)| - 3$ for any subgraph H of G with $E(H) \neq \emptyset$. A Laman graph has a property of being *minimally rigid* in the plane if it is realized as a *generic bar-joint framework* [5,4]. A bar-joint framework is a straight-line realization of a graph in the plane, and by regarding each edge as a bar and each point as a joint the rigidity of such a graph can be defined in a natural way (see, e.g., [4]). One of the most fundamental results in combinatorial rigidity theory asserts that a graph G realized on a generic point set (i.e., the set of the coordinates is algebraically independent over the rational field) is rigid if and only if G contains a spanning Laman subgraph [5]. Laman graphs appear in a wide range of applications, not only statics but also mechanical design such as linkages, design of CAD systems, analysis of protein flexibility, and sensor network localization [11,10].

⋆ Supported by ARC DP0881706 and ARC FT120100943.
⋆⋆ Supported by JSPS Grant-in-Aid for Scientific Research(A)(25240004).
⋆⋆⋆ Supported by NSC 97-2221-E-007-054-MY3 and NSC 100-2628-E-007-020-MY3.
† Supported by JSPS Grant-in-Aid for Scientific Research(A)(25240004).

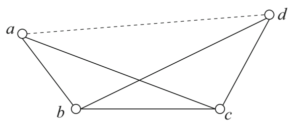


Fig. 1. $\mathsf{MLG}(P)$ that has an edge crossing

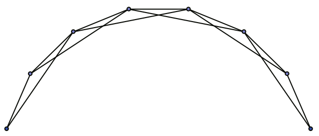


Fig. 2. $\mathsf{MLG}(P)$ that has $\Theta(|P|)$ edge crossings

Throughout the paper, by *a graph on a point set P* we mean a graph drawn in the plane with straight-line edges and vertex set P. In this paper we shall consider a bar-joint framework as a straight-line drawing in the plane, and we shall analyze geometric properties of the *Euclidean minimum-weight Laman graph* $\mathsf{MLG}(P)$ on a planar point set P, that is, a Laman graph on P with the minimum total edge-length over all Laman graphs on P. For simplicity of proofs, we assume throughout this paper that no three points in P are collinear and all interpoint distances are distinct. The point set satisfying these assumptions is called *semi-generic* in this paper.

A graph on a point set is called *plane* (or *non-crossing*) if two edges do not have an intersection except possibly at their endpoints. Extending the notion of planarity, a graph on a point set is called *k-plane* if each edge crosses at most k other edges [8]. Also, there is another extended notion, called *quasi-planarity*, where no three edges mutually cross each other except for their endpoints [1].

Our study is motivated by the well-known property of the Euclidean minimum spanning tree. For any semi-generic point set P on the plane, the Euclidean minimum spanning tree on P ($\mathsf{MST}(P)$ for short) is non-crossing. This is derived from the fact that $\mathsf{MST}(P)$ is a subset of the Delaunay triangulation of P, and this property brings us an important algorithmic consequence. Namely, $\mathsf{MST}(P)$ can be computed in $O(n \log n)$ time, which is faster than algorithms for general weighted graphs [9].

Observe that both Laman graphs and spanning trees are characterized by similar *sparsity conditions.* In general, a graph G is called *(k,l)-sparse* if $|E(H)| \leq k|V(H)| - l$ for any subgraph H of G with $E(H) \neq \emptyset$, and a (k,l)-sparse graph is called *(k,l)-tight* if it has exactly $k|V(G)| - l$ edges (see, e.g., [6]). A spanning tree is nothing but a $(1,1)$-tight graph while a Laman graph is a $(2,3)$-tight graph. (k,l)-sparse graphs have several common combinatorial properties such as being independent sets of a matroid. Hence a natural question is whether the Euclidean minimum-weight (k,l)-tight graph on a point set has a nice planarity property as does the Euclidean minimum-weight $(1,1)$-tight graphs in the plane. However it turns out that, unlike $\mathsf{MST}(P)$, $\mathsf{MLG}(P)$ may have a crossing in general (see Figure 1) and there is a point set P for which $\mathsf{MLG}(P)$ has $\Theta(|P|)$ crossings as shown in Figure 2.

One can describe the relation between $\mathsf{MST}(P)$ and the Delaunay triangulation in a detailed way by introducing the *the nearest neighbor graph*, *the relative neighborhood graph*, *the Gabriel graph* and *the Delaunay graph* [9]. To define these graphs, we introduce some notation, which will be used throughout the paper. For two points $p, q \in \mathbb{R}^2$, $\|pq\|$ denotes the Euclidean distance between p and q. We abuse the notation pq to stand for $\|pq\|$ when there is no confusion. In particular, we write $pq < rs$ if the length of segment pq is less than that of rs. The closed disk with the segment pq as diameter is denoted by D_{pq}. Also, for a point $p \in \mathbb{R}^2$ and $r \in \mathbb{R}$, the closed disk (resp. circle) with center p and radius r is denoted by $D_p(r)$ (resp. $C_p(r)$).

In the *$(k+1)$-nearest neighbor graph* $(k+1)$-$\mathsf{NNG}(P)$, an edge pq is included if and only if p is the i-th closest point among P from q for some $i \le k+1$ or vice versa. In the *k-relative neighborhood graph* k-$\mathsf{RNG}(P)$, pq is included if and only if $D_p(pq) \cap D_q(pq)$ contains at most k points among $P \setminus \{p, q\}$. In the *k-Gabriel graph* k-$\mathsf{GG}(P)$, pq is included if and only if D_{pq} contains at most k points among $P \setminus \{p, q\}$. In the *k-Delaunay graph* k-$\mathsf{DG}(P)$, pq is included if and only if there is a circle through p and q that contains at most k other points. As is well-known, 0-$\mathsf{DG}(P)$ is always a triangulation, called the Delaunay triangulation, if no four points lie on a circle. The following relations are classical (see for example [3,2,9]):

$$(k+1)\text{-}\mathsf{NNG}(P) \subseteq k\text{-}\mathsf{RNG}(P) \subseteq k\text{-}\mathsf{GG}(P) \subseteq k\text{-}\mathsf{DG}(P)$$

$$1\text{-}\mathsf{NNG}(P) \subseteq \mathsf{MST}(P) \subseteq 0\text{-}\mathsf{RNG}(P).$$

In this context, we prove the next relations. (The proof is given at the end of Section 2.)

Theorem 1. *Let P be a semi-generic set of points in the plane. Then*

$$\mathsf{MST}(P) \cup 2\text{-}\mathsf{NNG}(P) \subseteq \mathsf{MLG}(P) \subseteq 1\text{-}\mathsf{GG}(P) \cap 2\text{-}\mathsf{RNG}(P).$$

Ábrego et al. [2] recently investigated the crossing number and the maximum crossing number of proximity graphs, and they proved that k-$\mathsf{NNG}(P)$ has at most $k^3 n$ crossings for any P while there is a point set P such that k-$\mathsf{GG}(P)$ has $k^2n^2/4 + o(k^2n^2)$ crossings if $k = o(n)$. This result and Theorem 1 give rise to the following question: Does $\mathsf{MLG}(P)$ contain a linear number of crossings for every point set P? Our main theorem, proved in Section 4, affirmatively answers this question.

Theorem 2. *(6-planarity) Let P be a semi-generic set of points in the plane. For every edge $ab \in \mathsf{MLG}(P)$, the number of edges crossing ab is at most six.*

Moreover, we prove the following theorem in Section 3.

Theorem 3. *(Quasi-planarity Theorem) Let P be a semi-generic set of points in the plane. No three edges of $\mathsf{MLG}(P)$ pairwise cross.*

It is known that there is a point set P for which 1-$\mathsf{GG}(P)$ contains three edges that mutually cross each other [3, Figure 4].

2 Preliminaries

For two points a, b in the plane, let $\text{lens}(a, b) = D_a(ab) \cap D_b(ab)$. Note that by the semi-generic assumption made on the point set, for any two distinct points $a, b \in P$, there is no point on the boundary of $\text{lens}(a, b)$ other than a, b.

Let $K(P)$ be the complete graph on P. We sometimes abuse $K(P)$ to denote the edge set of $K(P)$. An edge set E in $K(P)$ is called (k, l)-sparse if it induces a (k, l)-sparse graph. It is known that the family of $(2, 3)$-sparse edge sets on a point set P forms the family of independent sets of the so-called 2-dimensional rigidity matroid [4] on $K(P)$. Among many properties of matroids, we shall use the following facts (see, e.g., [7, Section 1.8]).

Lemma 1. *For a point set P in the plane, $\mathsf{MLG}(P)$ can be computed by a greedy algorithm; it maintains a $(2, 3)$-sparse edge set I which is initially set to $\emptyset$, and considers an edge one by one in the nondecreasing order of their lengths, and adds it to I if the addition of the element to I maintains the $(2, 3)$-sparsity.*

Lemma 2. *Let P be a point set in the plane, $Q \subseteq P$, and $a, b \in Q$. Also let $E' = \{pq \in K(Q) \mid pq < ab\}$. If E' contains the edge set of a Laman graph on Q, then $ab \notin \mathsf{MLG}(P)$.*

Lemma 2 in particular implies that, for $Q \subseteq P$ with $|Q| = 4$, the longest edge in $K(Q)$ does not belong to $\mathsf{MLG}(P)$ since K_4 violates the $(2, 3)$-sparsity condition. This fact will be frequently used.

Lemma 3. *Let P be a semi-generic point set in the plane, and let $ab \in \mathsf{MLG(P)}$.*

(i) *Let R be one half of* $\text{lens}(a, b)$ *divided by the edge ab. Then there exists at most one point in the interior of R.*
(ii) *Suppose that there exists one point in each half of* $\text{lens}(a, b)$ *(say, p and q). Then $pq > ab$ holds, and $pq \notin \mathsf{MLG}(P)$.*

Proof. (i) Suppose that there exist two points (say, c, d) in one half of $\text{lens}(a, b)$. Then, consider the four-point set $Q = \{a, b, c, d\}$, and observe that ab is the longest edge in $K(Q)$. Therefore, Lemma 2 implies $ab \notin \mathsf{MLG}(P)$, a contradiction.

(ii) It follows from $p, q \in \text{lens}(a, b)$ that $\max\{ap, aq, bp, bq\} < ab$. Therefore, if $pq < ab$, $ab \notin \mathsf{MLG}(P)$ holds by Lemma 2. Since $ab \in \mathsf{MLG}(P)$, we have $pq > ab$, and hence $pq \notin \mathsf{MLG}(P)$ by Lemma 2. □

Proof of Theorem 1: Let us show $\mathsf{MST}(P) \subseteq \mathsf{MLG}(P)$. Consider any edge $ab \in \mathsf{MST}(P)$ and let $E' = \{pq \in K(P) \mid pq < ab\}$. Since $ab \in \mathsf{MST}(P)$, ab connects distinct connected components of the graph (P, E') on P. Suppose to the contrary that $ab \notin \mathsf{MLG}(P)$. Then there is $F \subseteq E'$ such that F satisfies the $(2, 3)$-sparsity condition but $F + ab$ violates the $(2, 3)$-sparsity condition. However, since ab is a bridge in the graph $(P, F + ab)$, it can be easily seen that a connected component of (P, F) violates the $(2, 3)$-sparsity condition, contradicting the $(2, 3)$-sparsity of F. Thus $\mathsf{MST}(P) \subseteq \mathsf{MLG}(P)$.

To see 2-NN$G(P) \subseteq \mathsf{MLG}(P)$, we first note the following well-known fact (e.g., [4]). Let F be an edge set and v be a vertex, and suppose that F contains exactly two edges e_1 and e_2 incident to v. Then F is $(2,3)$-sparse if and only if $F \setminus \{e_1, e_2\}$ is $(2,3)$-sparse.

Now we consider the greedy algorithm that computes $\mathsf{MLG}(P)$, and an arbitrary vertex $v \in V$ as well as the shortest and the second shortest edges e_1 and e_2 incident to v. Then the algorithm always accepts e_1 and e_2 because v has degree at most two when the algorithm tests the $(2,3)$-sparsity of the graph obtained by adding e_1 or e_2.

To see $\mathsf{MLG}(P) \subseteq$ 1-GG(P), suppose that D_{ab} contains two points (say, p, q) for some edge ab in $\mathsf{MLG}(P)$. Then, in $K(\{a,b,p,q\})$, ab is the longest edge which contradicts $ab \in \mathsf{MLG}(P)$ by Lemma 2. $\mathsf{MLG}(P) \subseteq$ 2-RNG(P) is immediate from Lemma 3(i). □

3 Quasi-planarity of MLG(P)

Given a line segment ab in the plane, the perpendicular bisector of ab (denoted by bisect(ab)) divides the plane into halfplanes R_a containing a and R_b containing b. Since P is semi-generic, all interpoint distances are distinct, for any distinct points $a, b \in P$, no other point in P is on the boundary of lens(a,b). In the proof of Theorem 3 we use the following property of six points in the plane.

Lemma 4. *Let $Q = \{a,b,c,d,e,f\}$ be a semi-generic set of six points in the plane, and suppose that the three line segments ab, cd and ef intersect each other. Then there exists at least one lens among* lens(a,b)*,* lens(c,d) *and* lens(e,f) *that contains two points among Q in its interior.*

Before proving Lemma 4, we show the following two lemmas.

Lemma 5. *Let $\{a,b,c,d\}$ be a semi-generic set of four points in the plane such that the two line segments ab and cd intersect each other. Suppose that $c, d \notin$* lens(a,b) *and segment cd cuts the boundary of* lens(a,b) *once in R_a and once in R_b. Then* lens(c,d) *contains both a and b in its interior.*

Proof. Assume without loss of generality that ab is a vertical segment, and that c (resp. d) belongs to R_a (resp. R_b) and lies on the right (resp. the left) halfplane delimited by the supporting line of ab. See Figure 3.

We first show that $ad < cd$. If the line segment ac does not intersect lens(a,b), bisect(ac) never intersects the left half of lens(a,b) and passes to the right of b, and thus $ad < cd$ holds. Therefore, we consider the case that ac intersects lens(a,b). Let c' be the intersection point of ac and lens(a,b). Then bisect(ac') is parallel to and on the left side of bisect(ac). Also, it passes through b, and thus bisect(ac) passes the right of b. Therefore, d lies on the left side of bisect(ac). Thus, $ad < cd$ holds.

Now let us consider bisect(ad). Let d' be the intersection point of ad and the boundary of lens(ab) and m be the midpoint of ab. Since bisect(ad') always

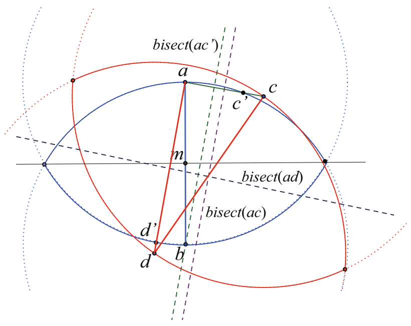


Fig. 3. Illustration of the proof of Lemma 5

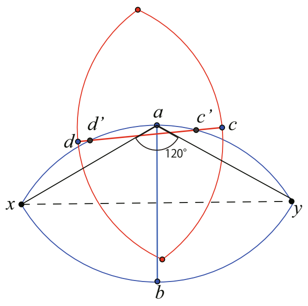


Fig. 4. Illustration of the proof of Lemma 6(i)

passes below m, so does bisect(ad). Thus, $ac < cd$ holds. Therefore, from $ad < cd$ and $ac < cd$, $a \in \text{lens}(c, d)$ follows.

Replacing a with b, we can show $b \in \text{lens}(c, d)$ by the same argument. This proves the lemma. □

For two line segments s and s', the *angle* between s and s' is defined to be the smaller angle (it is at most 90°) formed by the supporting lines of s and s'.

Lemma 6. *Let $\{a, b, c, d\}$ be a semi-generic set of four points such that the two line segments ab and cd intersect each other. Suppose that $c, d \notin \text{lens}(a, b)$, and that segment cd cuts the boundary of* lens(a, b) *twice in R_a. Then (i)* lens(c, d) *contains a, and (ii) the angle between ab and cd is greater than* 60°.

Proof. Let x, y be the intersection points of circles $C_a(ab)$ and $C_b(ab)$.

(i) Then, $\angle xay = 120°$ holds. Since c and d are separated by the supporting line of ab as shown in Figure 4, the angle $\angle cad > 120°$. This implies $a \in \text{lens}(c, d)$.

(ii) It is easy to see that the angle between xy and cd is less than 30°. Since xy is perpendicular to ab, the lemma follows. □

Proof of Lemma 4: Suppose to the contrary that none of the three lenses contains two points of $Q = \{a, b, c, d, e, f\}$ in its interior.

Let us first consider two lenses, lens(a, b) and lens(c, d). By Lemmas 5 and 6, if $c, d \notin \text{lens}(a, b)$, then $a \in \text{lens}(c, d)$ or $b \in \text{lens}(c, d)$ holds. This implies that at least one of $c \in \text{lens}(a, b)$, $d \in \text{lens}(a, b)$, $a \in \text{lens}(c, d)$ or $b \in \text{lens}(c, d)$ holds.

The corresponding relation holds for any pair of the three lenses. Since none of the three lenses contains two points of Q in its interior, without loss of generality, we may assume that $\text{lens}(a, b) \cap Q = \{a, b, c\}, \text{lens}(c, d) \cap Q = \{c, d, e\}$, and $\text{lens}(e, f) \cap Q = \{e, f, a\}$.

Note then that ab cuts the boundary of lens(c, d) twice in R_c or twice in R_d, since otherwise lens(c, d) contains a or b by Lemma 5. Therefore, the angle between ab and cd is greater than 60°.

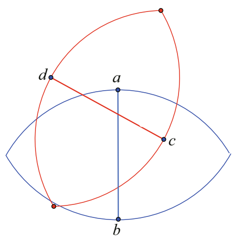


Fig. 5. Illustration of Case 1 in the proof of Lemma 4

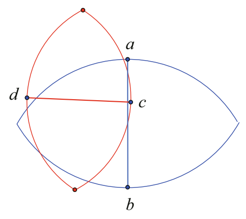


Fig. 6. Illustration of Case 2 in the proof of Lemma 4

By a symmetric argument, the angle between any pair of ab, cd and ef is greater than $60°$ (and at most $90°$ by definition). However, this leads to a contradiction because the sum of those three angles must be equal to $180°$ or the sum of two angles is equal to the other. $\square$

Proof of Theorem 3: Suppose to the contrary that three edges ab, cd, ef of $\mathsf{MLG}(P)$ intersect each other.

By Lemma 4, we can assume without loss of generality that lens(a, b) contains two points among c, d, e, f. By Lemma 3(ii) it cannot happen that both c and d belong to lens(a, b), and that both e and f belong to lens(a, b). Therefore, assume without loss of generality that $c, e \in$ lens(a, b) and $d, f \notin$ lens(a, b). Suppose that ab is a vertical line segment. Let l be the supporting line of ab, and L and R be the left halfplane and the right halfplane of l, respectively. Without loss of generality, we assume that c is on the right side of l. Then e is on the left side of l by Lemma 3(i). Also, since both cd and ef intersect ab, we have $d \in L$ and $f \in R$.

Let l_{ce} be the supporting line of bisect(ce), R_e be the halfplane determined by l_{ce} that contains e and R_c be the halfplane determined by l_{ce} that contains c. Observe then that $d \notin R_c$ or $f \notin R_e$ holds. To see this, suppose to the contrary that $d \in R_c$ and $f \in R_e$. Then we have $c, d \in R_c$ and $e, f \in R_e$ hold, and hence cd and ef do not intersect, a contradiction.

Therefore, without loss of generality, we may assume that $d \in R_e$. Next we show that c is the closest point from d among $\{a, b, c\}$. To see this, recall that $\max\{ac, cb, eb, ea\} < ab$. Moreover, since $d \in R_e$, we have $ed < cd$. Suppose that $ad < cd$. Then, we have $\max\{ac, cb, eb, ea, ed, ad\} < \max\{ab, cd\}$. Moreover observe that both $\{ac, cb, eb, ea, ed, ad, ab\}$ and $\{ac, cb, eb, ea, ed, ad, cd\}$ are Laman graphs on $\{a, b, c, d, e\}$, which means, by Lemma 2, that at least ab or cd does not exist in $\mathsf{MLG}(P)$, a contradiction. Therefore we have $ad > cd$.

The same argument also implies that $bd > cd$, and hence c is the closest point from d among $\{a, b, c\}$.

Let x be the leftmost point of lens(a, b) and α be the center of the circumcircle of triangle abc (see Figure 7(a)). Note that α is on the supporting line of bisect(ab). Since c is the closest point from d among $\{a, b, c\}$ and $d \in L \backslash$lens(a, b),

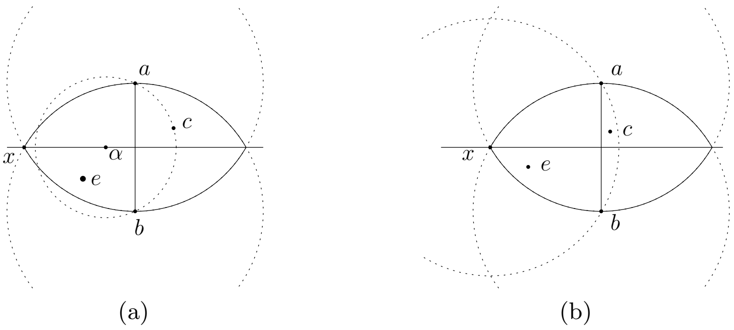


Fig. 7. Illustrations of the proof of Theorem 3

α is in the left side of x, which in particular implies that c is in $D_x(ab)$, the disk with center x of radius ab as shown in Figure 7(b).

Consequently, both c and e belong to $D_x(ab) \cap D_a(ab) \cap D_b(ab)$, and hence we obtain $ce < ab$. This in turn implies that ab is the longest edge among those on the four points $\{a, b, c, e\}$, and hence $ab \notin \mathsf{MLG}(P)$ by Lemma 2, a contradiction.

This completes the proof of Theorem 3. □

4 6-planarity of $\mathsf{MLG}(P)$

In this section, we prove the 6-planarity of $\mathsf{MLG}(P)$. Let us fix an edge $ab \in \mathsf{MLG}(P)$ which is assumed to be vertical. By Lemma 3(ii), edges crossing ab can be classified into two types. *Lens-crossing edges* are the edges whose both endpoints are outside of lens(ab). The other edges are called *fan-crossing edges* and they are the edges with exactly one endpoint inside lens(ab).

The proof of Theorem 2 consists of two parts.

(i) For an edge $ab \in \mathsf{MLG}(P)$, the number of lens-crossing edges is at most two (Lemma 7).

(ii) For an edge $ab \in \mathsf{MLG}(P)$, the number of fan-crossing edges is at most four (Lemma 10).

Lemma 7. *For any edge $ab \in \mathsf{MLG}(P)$, the number of lens-crossing edges is at most two.*

Proof. Let us assume that ab is a vertical segment. Notice that there is no lens-crossing edge that cuts the boundary of lens(ab) once in R_a and once in R_b by Lemma 5 and Lemma 3(ii). We prove that there exists at most one lens-crossing edge that cuts the boundary of lens(ab) twice in R_a, and there exists at most one lens-crossing edge that cuts the boundary of lens(ab) twice in R_b.

Suppose that there are two lens-crossing edges cd and ef, each of which cuts the boundary of lens(ab) twice in R_a. We first consider the case when $\{c, d, e, f\}$ are distinct points. Let x and y be intersection points of two circles $C_a(ab)$ and $C_b(ab)$. Let l and l' be the supporting lines of line segments xa and ya, respectively. Points c, d, e, f lie in the double wedge region delimited by two lines l and l'

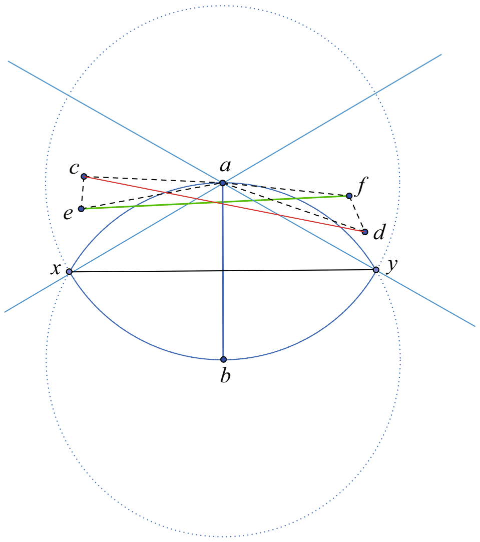


Fig. 8. Illustration of the proof of Lemma 7

(of course they are outside of lens(ab) by definition, see Figure 8). Since $\angle cae < 60°$, $ce < \max\{ac, ae\}$ holds. Similarly, since $\angle daf < 60°$, $df < \max\{ad, af\}$ holds. Also, since $\angle cad$ and $\angle eaf$ are larger than $120°$, $ac < cd$, $ad < cd$, $ae < ef$ and $af < ef$ hold. Therefore, $\max\{ac, ae, ce, ad, af, df\} < \max\{cd, ef\}$. Since $\{ac, ae, ce, ad, af, df, cd\}$ and $\{ac, ae, ce, ad, af, df, ef\}$ are both Laman graphs on the five-point set $Q = \{a, c, d, e, f\}$, it cannot happen that both cd and ef belong to $\mathsf{MLG}(P)$ by Lemma 2, a contradiction.

If $c = e$ or $d = f$ holds, say $c = e$, then it can be seen by the same argument that $\max\{ac, ad, af, df\} < \max\{cd, ef\}$, which implies by Lemma 2 that $\mathsf{MLG}(P)$ does not contain cd or ef, a contradiction.

This proves the lemma. □

Now we consider the number of fan-crossing edges. Lemma 10 is proved by showing the following two lemmas. (The proofs are omitted.)

Lemma 8. *Let c be a point in* lens(a, b). *Then, there exist at most four edges emanating from c that cross ab.*

Therefore there exist at most eight fan edges that cross ab, four edges are emanating from a point in the right half and the other four from a point in the left half lens. The following lemma improves this bound.

Lemma 9. *Let c (resp. d) be a point in the right half (resp. the left half) of* lens(a, b) *such that there exists at least one fan-crossing edge emanating from c (resp. d). Then there exist at most two fan edges emanating from c (resp. d) that cross ab, respectively.*

From Lemmas 8 and 9, we have the following.

Lemma 10. *For each edge ab of* $\mathsf{MLG}(P)$*, there exist at most four fan-crossing edges that cross ab.*

By Lemma 7 and Lemma 10, we establish the main theorem, Theorem 2.
Finally, we show that the 6-planarity is tight.

Theorem 4. *There is a configuration P of 9 points in the plane such that an edge of* $\mathsf{MLG}(P)$ *is crossed by 6 edges of* $\mathsf{MLG}(P)$.

Proof. Consider the following example on $P = \{a, a', b, c, c', d, d', e, e'\}$ as shown in Figure 9. Let h be a large number (defined below). Parameter $\varepsilon > 0$ is chosen such that c lies outside of lens(a, a'), i.e. $a'c > 2h$. Note that $\lim_{h\to\infty} \varepsilon = 0$. Parameter $\delta > 0$ is chosen such that cc' is slightly larger than $2h$. We can assume that $\lim_{h\to\infty} \delta = 0$.

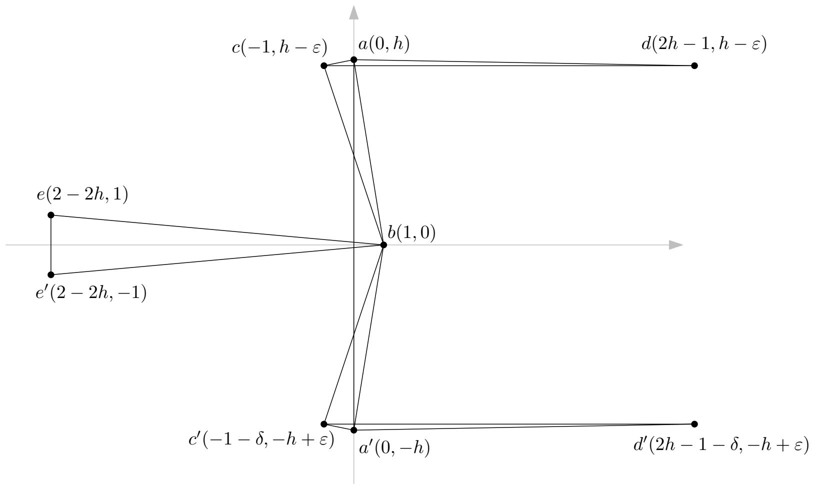


Fig. 9. Point set P and graph G for the proof of tightness of 6-planarity of $\mathsf{MLG}(P)$

Consider graph $G = (V, E)$ shown in Figure 9. Let E' be the set of pairs $x, y \in P$ such that $xy \le 2h$. We claim that, if h is sufficiently large, then $E' = E$. Indeed, the distances aa', cd and $c'd'$ are equal to $2h$. If h is large then $eb^2 = (2h-1)^2 + 1 < 4h^2$ and thus, $(e, b) \in E'$. By symmetry $e'b < 2h$. We show that $ec > 2h$. If h is large then

$$ec^2 = (2h-3)^2 + (h-1-\varepsilon)^2 > 4h^2.$$

Similarly $e'c' > 2h$ and $db, d'b > 2h$. Note that $dd' = cc' > 2h$. Therefore $E' = E$. By Lemma 1 graph G is a subgraph $\mathsf{MLG}(P)$ since E is $(2, 3)$-sparse. Then $\mathsf{MLG}(P)$ contains edge aa' which is crossed 6 times. □

Applying the similar argument as Lemma 7 and Lemma 8, it is not difficult to prove a bound of $O(k)$ for the number of crossings per edge in the Euclidean minimum weight (k, l)-tight graph on P, but we omit the detail since all the ideas are already presented in the case of $k = 2$ and $l = 3$. Improving the hidden constant is left as a future work.

Acknowledgment. This research was partially supported by the University of Sydney through a grant from the International Program Development Fund.

References

1. Agarwal, P.K., Aronov, B., Pach, J., Pollack, R., Sharir, M.: Quasi-planar graphs have a linear number of edges. Combinatorica 17(1), 1–9 (1997)
2. Ábrego, B.M., Fabila-Monroy, R., Fernández-Merchant, S., Flores-Peñaloza, D., Hurtado, F., Sacristán, V., Saumell, M.: On crossing numbers of geometric proximity graphs. Computational Geometry: Theory and Applications 44(4), 216–233 (2011)
3. Bose, P., Collette, S., Hurtado, F., Korman, M., Langerman, S., Sacristan, V., Saumell, M.: Some properties of k-Delaunay and k-gabriel graphs. Computational Geometry: Theory and Applications 46(2), 13–16 (2013)
4. Graver, J., Servatius, B., Servatius, H.: Combinatorial Rigidity. Graduate Studies in Mathematics, vol. 2. American Mathematical Society (1993)
5. Laman, G.: On graphs and rigidity of plane skeletal structures. Journal of Engineering Mathematics 4, 331–340 (1970)
6. Lee, A., Streinu, I.: Pebble game algorithms and sparse graphs. Discrete Mathematics 308, 1425–1437 (2008)
7. Oxley, J.: Matroid Theory, 2nd edn. Oxford Graduate Texts in Mathematics, vol. 21. Oxford University Press (2011)
8. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. Combinatorica 17(3), 427–439 (1997)
9. Preparata, F.P., Shamos, M.I.: Computational Geometry - An Introduction. Springer (1985)
10. Servatius, B.: The geometry of frameworks: rigidity, mechcanism and CAD. In: Gorini, C.A. (ed.) Geometry at Work: A Collection of Papers Showing Applications of Geometry. Cambridge University Press (2000)
11. Thorpe, M.F., Duxbury, P.M. (eds.): Rigidity Theory and Applications. Kluwer Academic/Plenum Publishers, New York (1999)

Structure and Computation of Straight Skeletons in 3-Space[*]

Franz Aurenhammer and Gernot Walzl

Institute for Theoretical Computer Science,
Graz University of Technology, Graz, Austria
{auren,gernot.walzl}@igi.tugraz.at

Abstract. We characterize the self-parallel (mitered) offsets of a general nonconvex polytope $\mathcal{Q}$ in 3-space and give a canonical algorithm that constructs a straight skeleton for $\mathcal{Q}$.

1 Introduction

The *straight skeleton* of a polygon P in the plane is a versatile skeletal structure, composed of angular bisectors of P and thus of piecewise linear components [1,8,12]. A well-known procedural definition exists, by a self-parallel shrinking process for P and the resulting 'events' that construct the skeleton nodes. Events are unique changes in the polygon boundary, yielding the *mitered offset* of P, in contradistinction to the Minkowski sum offset specified by the medial axis of P; see e.g. [7,11]. Applications arise in diverse areas, including computer graphics, robotics, architecture, and geographical information systems.

To construct a straight skeleton for a given polytope $\mathcal{Q}$ in 3-space, one tries to proceed in a way analogous to the planar case. The facet planes of $\mathcal{Q}$ are offset simultaneously, self-parallel, and at unit speed. Thereby, the shrinking polytope undergoes changes of geometric, combinatorial, and topological nature. Geometrical changes, of course, take place continuously, whereas combinatorial changes (on the polytope boundary) and topological changes (like new tunnels, or breaking the polytope locally or globally apart) occur once in a while. Each type of change implies the former ones.

During the offsetting process, the edges and vertices of $\mathcal{Q}$ trace out the facets and edges, respectively, of the spatial skeleton. Thus a concise specification of the offsetting process results in a concise definition of the skeleton. However, unlike parallel offsets of polygons, parallel offsets of polytopes in $\mathbb{R}^3$ are in general not unique. This might be among the reasons why not much literature exists on this topic. Barequet et al. [4] studied straight skeletons in 3-space, mainly for the special case of orthogonal (i.e., axes-aligned) polytopes, where the skeleton is the medial axis of the polytope for the L_∞-metric (and thus can be defined via distances). They mention ambiguity problems for the case of general polytopes, referring to Demaine et al. [6]. Also, a nice offsetting possibility by means of

[*] Supported by ESF Programme EuroGIGA - Voronoi, Austrian Science Foundation.

the planar weighted straight skeleton [2] is observed, but the offset treatment of general vertices remains unclear (as is explained in Subsection 2.2). In fact, it remains open whether a boundary-continuous and non-selfintersecting offset polytope always has to exist. For orthogonal polytopes, Martinez et al. [10] recently described an implementation of a straight skeleton construction algorithm that uses the space-sweep technique.

In this paper we give a systematic treatment of offsetting, in terms of generalized *vertex figures* of a given polytope $\mathcal{Q}$ in $\mathbb{R}^3$, and their spherical *bisector graphs* (Subsection 2.1). This includes a characterization of all possible offsets for $\mathcal{Q}$, and consequently, of all possible spatial straight skeletons for $\mathcal{Q}$.

As an algorithmic tool, the so-called *spherical skeleton* is introduced (Subsection 2.2), a generalization to the unit sphere of the classical unweighted straight skeleton in the plane. This structure can uniformly treat arbitrary polytope vertices, and all types of events that occur during the construction of a straight skeleton in $\mathbb{R}^3$, including those which change the topology of the polytope. A canonical offsetting algorithm results (Section 3), which produces a small number of edges, among all offset possibilities, and comes with a certain optimality property concerning edge convexity. The algorithm also provides a means for converting boundary-triangulated (*simplicial*) nonconvex polytopes in $\mathbb{R}^3$ into 3-regular (*simple*) ones, via ε-thinning; see e.g. [11].

We define a *polytope* $\mathcal{Q}$ in 3-space $\mathbb{R}^3$ as a bounded, closed, and interior-connected subset of $\mathbb{R}^3$ with piecewise linear boundary. The boundary components of $\mathcal{Q}$ of dimensions 2, 1, and 0, respectively, are called *facets*, *edges*, and *vertices* of $\mathcal{Q}$. In the easiest case, $\mathcal{Q}$ is homeomorphic to a ball in $\mathbb{R}^3$.

A polytope is, in general, nonconvex and may contain tunnels, and even 'voids' that make its boundary disconnected. However, if the input polytope $\mathcal{Q}$ is boundary-connected, no holes can be created in its offsetting process.

An edge e of $\mathcal{Q}$ is called *reflex* if there exists some line segment $\ell \subset \mathcal{Q}$ whose interior intersects the interior of e in a single point. Otherwise, edge e is called *convex.* (Topological notion like interior, boundary, etc. is meant relative to the dimension of an object.)

2 Vertex Figure Resolution

2.1 Offset Characterization

In the offsetting process, the local facial structure of the input polytope has to be updated, at existing or arising vertices of degree 4 or higher (apart from certain degenerate cases). We describe these changes in terms of so-called vertex figures of a polytope [9].

Consider some vertex, v, of the current polytope $\mathcal{Q}$. Center a sphere at v, sufficiently small to intersect only faces of $\mathcal{Q}$ incident to v. We can assume that, w.l.o.g., this sphere is the unit sphere, U. The *vertex figure*, $\mathcal{F}(v)$, of v is then defined as the intersection of $\mathcal{Q}$ with the unit ball; see Figure 1 (left).

The boundary of $\mathcal{Q}$ intersects U in a Jordan curve $\mathcal{J}$ composed of great arc segments on U. (For ease of exposition, assume only one such curve for now.)

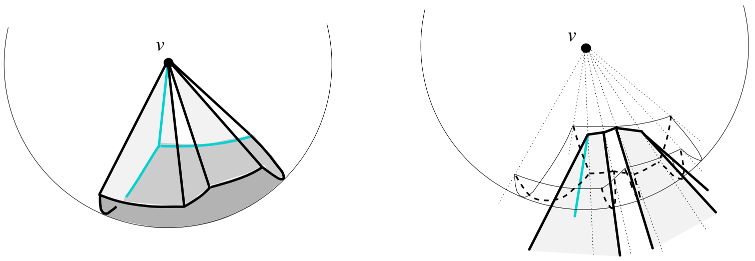


Fig. 1. Vertex figure and spherical polygon $\mathcal{P}$ (left). Offsetting the vertex figure (right). The bisector graph G (dashed style) used for projecting is unique for $\mathcal{P}$ in this example. The degree-6 vertex v splits into four vertices of degree 3 in the offset surface $\Gamma(G)$.

Then $\mathcal{Q} \cap U$ is a simply connected spherical polygon, $\mathcal{P}$. The curve $\mathcal{J}$ consists of *nodes* and *segments*; the former are the intersections of U with the edges of $\mathcal{Q}$ incident to v, and the latter belong to the boundary of the planar facets $f_1, \ldots, f_m$ of the vertex figure $\mathcal{F}(v)$. Here $m \geq 3$ is the degree of vertex v.

Let now H_i be the supporting plane of facet f_i. Denote with H_i^{Δ} the parallel offset of H_i by $\Delta > 0$, inward with respect to $\mathcal{F}(v)$. An *(inward) offset* of $\mathcal{F}(v)$ is defined as a radially monotone (w.r.t. v) and continuous surface over $\mathcal{P}$ with facets from the planes H_i^{Δ}, including an unbounded facet for each plane.

For increasing Δ, the intersection line $\ell_{ij} = H_i^{\Delta} \cap H_j^{\Delta}$ of two offset planes sweeps along an angular bisector plane, B_{ij}, of H_i and H_j. The point of intersection of three offset planes moves along the trisector line of these planes. Note that these bisector planes and trisector lines all pass through the vertex v.

We next consider the picture on the unit sphere U. Define $b_{ij} = B_{ij} \cap U$, which is a great circle on U. Exactly three great circles b_{ij}, b_{ik}, b_{jk} meet in two common points, which are the intersections of U with the respective trisector line. (We only discuss the non-degenerate situation here.) For a system of circles with this property, we can consider any *bisector graph* G they define inside the spherical polygon $\mathcal{P}$.

G is defined as a crossing-free graph with labelled arcs $a_{ij} \subseteq b_{ij}$, where the ordering of the labels (ij) indicates the position of the facets f_i and f_j with respect to the bisector plane B_{ij}. G contains nodes of degree 1 (the nodes of $\mathcal{P}$), and of degree 3 whose incident arcs have labels of the form (ij), (ik), (kj). G can be disconnected and can even contain cycles. We have the following characterization.

Lemma 1. *For every offset surface for $\mathcal{F}(v)$ its edge graph maps, by central projection with respect to v, to some bisector graph for the system $(b_{ij})_{1 \leq i < j \leq m}$. Conversely, each bisector graph G for this system lifts, by this projection, to the edge graph of an offset surface $\Gamma^{\Delta}(G)$ for $\mathcal{F}(v)$, which is unique for fixed $\Delta > 0$.*

Proof. From the way how offset planes intersect, it is obvious that each valid offset surface radially projects to a crossing-free degree-3 graph with the required labelling. To prove the converse, we lift G's arcs a_{ij} to the lines $\ell_{ij} = H_i^{\Delta} \cap H_j^{\Delta}$ by central projection with respect to v. This can be done because a_{ij}, ℓ_{ij}, and v

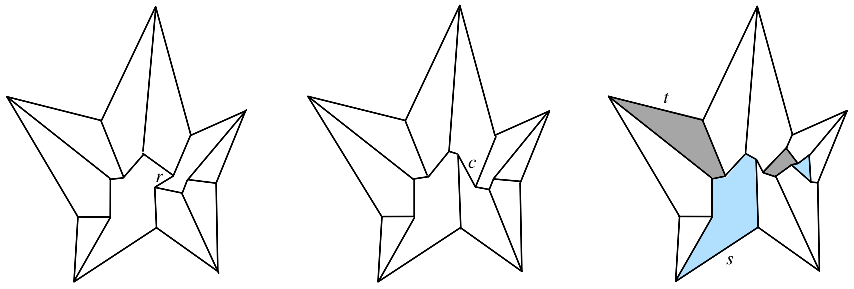


Fig. 2. Different offset surfaces for a vertex figure of degree 10. The figure is based on a pentagonal star $\mathcal{P}$ (of small area, so that $\mathcal{P}$ is almost flat.) Our algorithm will produce the offset with the convex edge c (middle), avoiding the reflex edge r (left). The bisector graph is highly ambiguous and even need not be a tree. Additional small facets for two segments s and t bounding $\mathcal{P}$ might show up in the surface, either together (right), or separately (not shown).

are contained in the same plane, B_{ij}. Each connected face of G thus lifts to a polygon in 3-space, which is planar because its edges e_{ij} are labelled with the same offset plane index j on the 'inside', by the labelling of G. That is, G lifts to a unique piecewise linear surface, whose facets fit continuously because their edges e_{ij} are part of ℓ_{ij} and thus lie in both offset planes H_i^{Δ} and H_j^{Δ}. The surface is radially monotone because G is crossing-free.

See Figure 1 (right) for an illustration. The combinatorial structure of $\Gamma^{\Delta}(G)$ does not change for $\Delta > 0$, so we may just write $\Gamma(G)$.

In general, there is more than one valid bisector graph for a given system (b_{ij}). In fact, there can be exponentially many, in the degree m of v (which can be obtained, for example, by combining the graphs shown in Figure 2 into bigger ones). Consequently, there may be more than one valid offset surface for a given vertex figure. That is, parallel offsets of polytopes in $\mathbb{R}^3$, and with it, their straight skeletons, are not unique.

2.2 Spherical Skeleton

By Lemma 1, offsets of vertex figures – and in fact, of the entire polytope Q – can be computed by generating bisector graphs. In principle, any valid bisector graph for a spherical polygon $\mathcal{P}$ could be extracted from the arrangement of the great circles b_{ij} defined on the unit sphere U. A more direct and also unique way is to construct the *spherical skeleton* of $\mathcal{P}$, denoted by $\mathrm{Sph}(\mathcal{P})$. We now define this skeleton, by describing its events (and peculiarities).

We simulate the offsetting process, by considering the intersection of a vertex figure's facets with U. That is, we offset the Jordan curve $\mathcal{J}$ that bounds $\mathcal{P}$, inwards with respect to $\mathcal{P}$. The moving curve is denoted by $\mathcal{J}^{\Delta}$, where Δ means offset distance, hence time. (The case 'outwards' is analogous, and the case 'more than one Jordan curve' is similar.)

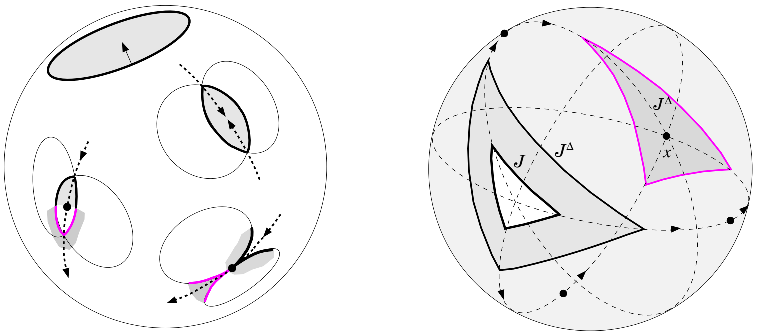


Fig. 3. Void event and arc merge event (left upper). Two stopping events; the interior of the offsetting region is shaded (left lower). A triangular Jordan curve $\mathcal{J}$ and two offsets $\mathcal{J}^\Delta$, for $\Delta < 1$ and $\Delta > 1$ (right). The three extending skeleton arcs (dashed) stop once each, at the marked positions. They meet at point x where $\mathcal{J}^\Delta$ vanishes in a triangle collapse.

$\mathcal{J}^\Delta$'s nodes move on the great circles b_{ij} and draw out the skeleton arcs. The moving nodes bound shifted segments s_i from $\mathcal{J}$ on circles $H_i^\Delta \cap U$ whose radii all shrink identically. The speed of an expanding skeleton arc a_{ij} is given by the (fixed) angle between the two planes H_i^Δ and H_j^Δ and by the time Δ. As one possible event, three such arcs a_{ij}, a_{jk}, and a_{ik} meet at the same time at a point of intersection of U with the respective trisector line. This means a *segment collapse* for s_j, making $\mathcal{J}^\Delta$ lose this segment. Or, one of the arcs, say a_{ij}, bumps into some shifting segment s_k and splits it (*segment split* event). This breaks $\mathcal{J}^\Delta$ into two closed curves, and lets two arcs a_{ik} and a_{jk} start at the breaking point. In both cases, a new node of the skeleton $\mathrm{Sph}(\mathcal{P})$ is created, quite similar to the planar skeleton case [1]. Also similar is the simultaneous collapse of three segments forming a spherical 3-gon (*triangle collapse*). On the other hand there are new events, for example the collapse of a spherical 2-gon (*arc merge*) or of a spherical 1-gon, i.e., a full circle (*void event*); see Figure 3 (left). These events do not create nodes of $\mathrm{Sph}(\mathcal{P})$.

If the vertex figure $\mathcal{F}(v)$ based on $\mathcal{P}$ is *pointed*, that is, if there exists a plane through v that has $\mathcal{P}$ on a fixed side, then the construction of $\mathrm{Sph}(\mathcal{P})$ will be completed at time $\Delta < 1$, when all the planes H_i^Δ are still intersecting the sphere U. Otherwise, more substantial differences to the planar case occur: At time $\Delta = 1$, the planes H_i^Δ will start avoiding U, although $\mathrm{Sph}(\mathcal{P})$ is still incomplete. To continue the construction, we let the planes return and intersect U in circles with negative radii, which are expanding now. (That is, the *complements* of the disks they bound are shrinking now, rather than the disks themselves.) Arc extension along b_{ij}, which stopped at time $1 - \Delta_{ij}$ and at point p_{ij} when and where the line $\ell_{ij} = H_i^\Delta \cap H_j^\Delta$ left U, continues after this *stopping event* at p_{ij} at time $1 + \Delta_{ij}$, with the arc endpoint carrying two segments $\overline{s_i}$ and $\overline{s_j}$ of mirrored shape which stem from negative circles; see Figure 3 (left lower). The convexity status of their common endpoint alters thereby.

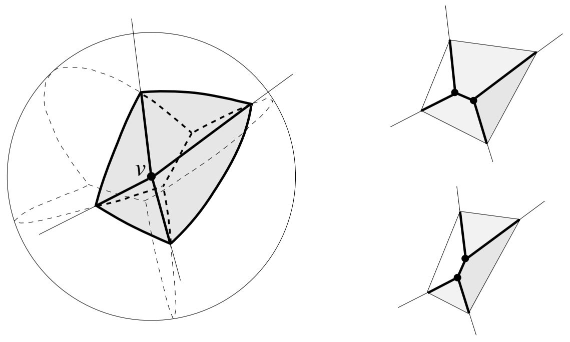


Fig. 4. Vertex touch; the convex case. A polytope edge shrinks to length zero, creating a vertex figure $\mathcal{F}(v)$ with four convex edges (left). Its base is a spherical rectangle $\mathcal{P}$. The two longest segments of $\mathcal{P}$'s boundary curve $\mathcal{J}$ yield an arc in the spherical skeleton (dashed style, inside $\mathcal{P}$) that connects two nodes. Accordingly, vertex v splits into two offset vertices after the event (right lower).
For the inverse event, the outward offset is relevant. The spherical skeleton now lives in the outer hemisphere $U \setminus \mathcal{P}$, which corresponds to the complement of $\mathcal{F}(v)$. Four skeleton arcs extend from $\mathcal{J}$ around U, but stop in between this time. The two shortest segments of $\mathcal{J}$ yield a skeleton arc now. Vertex v splits into two vertices again, yielding different facet adjacencies that describe the situation before the event (right upper).

(Intuitively speaking, the returning planes collect up their stopped arc endpoints in reverse order, letting them continue at the right time.)

This 'positive/negative switch' at time $\Delta = 1$ ensures that, for each trisector line, both intersection points with U are taken into account as possible skeleton nodes; see Figure 3 (right). For each stopped arc endpoint, its two incident segments are temporarily inactive for split events.

Sph($\mathcal{P}$) is a bisector graph which is outerplanar. Each segment s_i of $\mathcal{J}$ sweeps out a single region, which is incident to s_i. (Segments on the same great circle might give the same region.) For, once all segments for an offsetting circle $H_i^{\Delta} \cap U$ have collapsed (unless in a stopping event), the plane H_i^{Δ} cannot define any further piece of Sph($\mathcal{P}$). Thus Sph($\mathcal{P}$) contains $\leq m$ regions and $O(m)$ arcs and nodes, where m is the number of segments of $\mathcal{J}$. Figures 4 to 9 give illustrations.

Using *weighted* planar straight skeletons [2] to resolve high-degree vertices has been considered in Barequet et al. [4]. In their approach, partial skeletons in two osculating planes need to be merged into one skeleton. Thereby, components from one plane might destroy the skeleton structure in the other, however. For example, skeleton faces can expand, rather than shrink as they do for Voronoi-like partitions. It remains unclear how to deal with this problem, which is equivalent to the (unsolved) problem of computing a planar straight skeleton by divide & conquer, or incremental insertion.

To base the offsetting process on Sph($\mathcal{P}$) is natural, as it is consistent with the well-known process that constructs the straight skeleton in $\mathbb{R}^2$. This choice is also preferable, not least because of the linear size of Sph($\mathcal{P}$). For general bisector graphs, examples with $\Omega(m^2)$ interior faces can be constructed. Also,

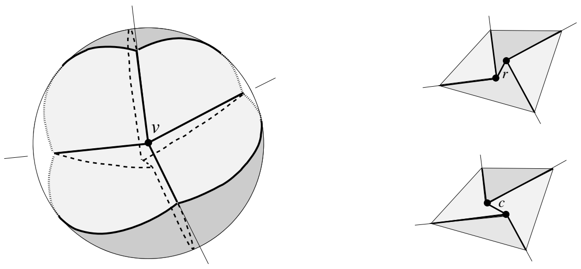


Fig. 5. Vertex touch; saddle point. $\mathcal{F}(v)$ has four edges, of alternating convexity type (left). Unlike the convex case in Fig. 4, two offsets exist (right). The spherical skeleton produces the lower solution. The short skeleton arc corresponds to a convex edge c in the offset polytope. It connects the two vertices which v is split into. In the upper solution, the offset edge r is reflex.

Sph($\mathcal{P}$) is unweighted, and thus behaves like the classical planar straight skeleton in certain respects. For example, if $\mathcal{F}(v)$ is pointed then one can show that no new *reflex* edges are generated in the offset surface $\Gamma(\mathrm{Sph}(\mathcal{P}))$, only the 'forced' ones that are also present in $\mathcal{F}(v)$. A similar property holds for the 'roof' of the planar straight skeleton [1], but not for weighted straight skeletons, in general.

Sph($\mathcal{P}$) can be computed in $O(m^2 \log m)$ time with a trivial implementation, like in the planar case [1]. Note that m (the degree of the vertex figure) will be usually quite small in practice, for two reasons: Most solids can be accurately approximated by polytopes having vertices of small constant degree. Also, in the non-degenerate case, the straight skeleton events in $\mathbb{R}^3$ lead to vertex figures of degree at most 8, as we shall see in the following section.

3 Straight Skeleton Algorithm in $\mathbb{R}^3$

When attempting to construct a straight skeleton in $\mathbb{R}^3$, the vertex figure resolution problem arises immediately, when vertices of degree $m \geq 4$ of the input polytope $\mathcal{Q}$ have to be split. In the non-degenerate case, such a vertex splits into $m - 2$ vertices of degree 3 in the shrunken polytope. Interestingly, by nature of the straight skeleton events in $\mathbb{R}^3$, the vertex resolution problem is encountered again each time a new event is to be handled.

An *event* in this sense is an increase in the number of facets for some vertex of the shrinking polytope (for example, by touch with another degree-3 vertex). We are then left with the problem of resolving that vertex, to continue the skeleton construction for $\mathcal{Q}$. As we have seen in Subsection 2.1, there are several possibilities to proceed. This gives a tree of possible offset polytopes, with root $\mathcal{Q}$. Each path from $\mathcal{Q}$ to a tree leaf corresponds to a different straight skeleton for $\mathcal{Q}$. (If $\mathcal{Q}$ is *convex* then there is only one path, leading to the *medial axis* of $\mathcal{Q}$.)

When the spherical skeleton approach from Subsection 2.2 is used, we always obtain a unique path. This path is short because offsets with a small number of edges are created, especially in the beginning where high-degree vertices of $\mathcal{Q}$

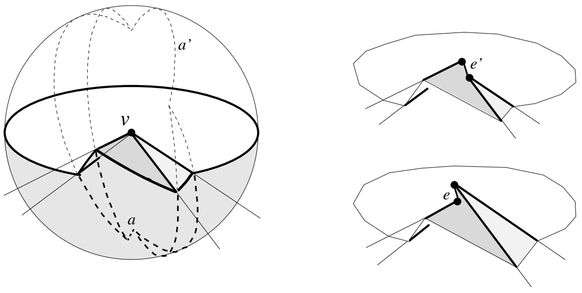


Fig. 6. Vertex touch; pointed saddle point. Again, $\mathcal{F}(v)$ has two reflex and two convex edges, but they do not positively span $\mathbb{R}^3$ as in Fig. 5. The skeleton lives in the lower hemisphere. It is generated by a segment collapse and a triangle collapse, followed by a void event at the south pole. The short skeleton arc, a, gives a convex edge e in the offset polytope (right lower), where the horizontal facet and the dark-shaded facet are adjacent. The same adjacency existed already as edge e' in the polytope before the event (right upper), indicated by arc a' in the skeleton in the upper hemisphere, which specifies the inverse event. Observe that a and a' lie on the same great circle.

may be present. The skeleton $\mathrm{Sph}(\mathcal{P})$ does not only encode the local facial structure at a vertex v after the event, but can also restore this structure *before* the event (except for certain collapse events, see later). To this end, the *outward* offset of the vertex figure $\mathcal{F}(v)$ based on $\mathcal{P}$ is considered. This is just the (inward) offset of the complement of $\mathcal{F}(v)$. We obtain it by using the spherical skeleton $\mathrm{Sph}(U \setminus \mathcal{P})$, as is illustrated in Figures 4, 6, 7, 8, and 9. In this way, an event can be associated with a unique *inverse invent.*

As a consequence, the resulting unique straight skeleton, SKEL(Q), for the polytope $\mathcal{Q}$ is given locally at each of its vertices v by

$$\mathrm{SKEL}(Q) \cap U = \mathrm{Sph}(\mathcal{P}) \cup \mathrm{Sph}(U \setminus \mathcal{P}) \ .$$

Our vertex resolution procedure works, without conceptual changes, for all occurring degenerate cases, be they by the special structure of $\mathcal{Q}$ (for example, regular pyramid vertices that offset into vertices of degree ≥ 4) or by construction (during event handling, where coplanarities may arise naturally).

3.1 Event Categorization

Categorizing the (non-degenerate) events that construct SKEL(Q) is relatively easy. A general distinction is between *surface events* which change the combinatorial structure of $\mathcal{Q}$'s edge graph without topological effect, and *solid events* which also change the topology of $\mathcal{Q}$. For the former, the vertex figure yields a single Jordan curve on U, whereas it gives two disjoint curves for the latter.

Note that a categorization of all possible events is *not* needed for handling them; our algorithm automatically computes a unique local offset structure.

In the generic case, all vertices of the offsetting polytope are of degree 3 before each event (except initially), and faces of dimension ≥ 1 touch only in their interiors

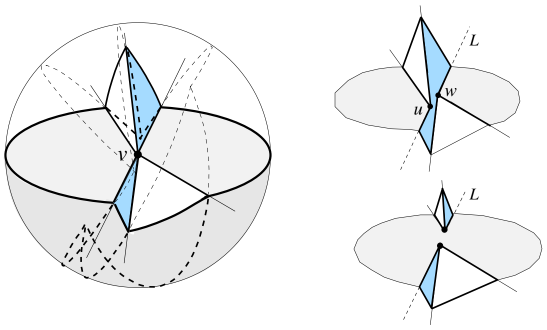


Fig. 7. Vertex touch; degree 6. In $\mathcal{F}(v)$, the two dark-shaded facets and the two horizontal facets are coplanar, respectively (left). Thus the coincidence of the two approaching polytope vertices u and w (right upper) that touch in v is not by degeneracy but by construction; they move on the same straight line L. Vertex v splits into two degree-3 vertices after the event (right lower), as is witnessed by two nodes of the same degree in the skeleton in the lower hemisphere. This skeleton is disconnected, as is the one for the inverse event in the complementary hemisphere.

in an event. Then a surface event is either a *vertex touch* (of one of various types displayed in Figures 4 to 7) which give a vertex figure $\mathcal{F}(v)$ of degree at most 6, a *vertex/edge touch* (like in Figure 8) where $\mathcal{F}(v)$ is of degree 5, or an *edge/edge touch* where $\mathcal{F}(v)$ is of degree 8. Solid events include the *piercing event* (vertex/facet touch, Figure 9), the *kissing event* (edge/edge touch), and the *splitting event*. For each of them, $\mathcal{F}(v)$ has 4 facets. *Tetrahedron collapse* is the last one (and also the chronologically final event), with 4 facets shrunk to a common point.

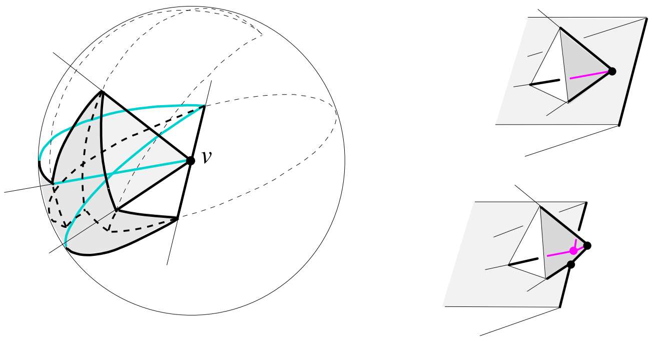


Fig. 8. Vertex/edge touch. The flat wedge retracts to the left, faster than does the triangular pyramid on its top (right upper). At v, the pyramid splits the rim of the wedge, whose lower facet then expands to above the rim in the offset polytope (right lower). Correspondingly, the degree-5 vertex v in $\mathcal{F}(v)$ splits into three vertices, as is witnessed by three nodes in the skeleton (left). This bisector graph for $\mathcal{F}(v)$ is unique. Therefore no other offsets are possible. (The case where the upper (horizontal) wedge facet expands at v leads to a wrong orientation of this plane. The corresponding graph leaves $\mathcal{P}$.) The situation before the event is encoded in the complementary skeleton, which is disconnected. Its single arc represents the unsplit rim.

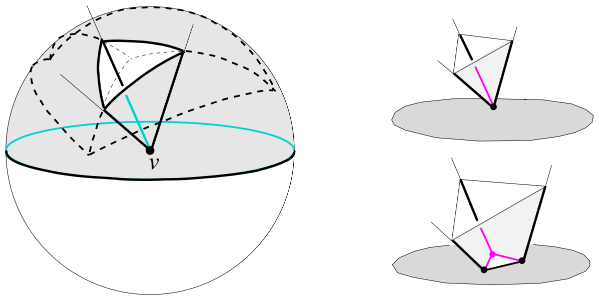


Fig. 9. Piercing event. $\mathcal{F}(v)$ is based on a 'spherical polygon' bounded by *two* Jordan curves, a spherical triangle and a full circle (left). The skeleton contains a triangular cycle that corresponds to the hole pierced into the horizontal facet by the offsetting pyramidal pit (right lower). Thus vertex v splits into three vertices.
For the inverse of the piercing event, the skeleton is just a degree-3 star inside the triangular Jordan curve, corresponding to the pyramid whose apex now rises above the horizontal polytope facet (right upper). The other curve, the full circle, contracts at the south pole without leaving a trace – a void event.

The anatomy of events can be quite complex. We therefore provided examples for most of them (but not all, due to space constraints), with detailed explanations in the figures' legends. The meaning of the events for the skeleton SKEL(Q) in $\mathbb{R}^3$ being constructed is briefly commented below.

Each event constructs a particular vertex v of SKEL(Q). The surface events in Figures 4, 5, and 6 complete a facet of SKEL(Q) at v, and start a new one. Adjacency of skeleton cells 'flips' in the first two cases but, interestingly, remains in the last case. For Figure 7 even the same facet, swept over by line L, continues at v, with two parts. Two new facets, traced out by small pyramid edges, start for Figure 8. Three new facets are created at v in a piercing event (Figure 9). This event yields a tunnel in the shrinking polytope.

4 Concluding Remarks

Concerning the size of SKEL(Q), each event implies that four offset planes meet at the same point, which happens only once during the entire offsetting process. This bounds the number of events by $\binom{n}{4}$, if $\mathcal{Q}$ has n vertices. The size of SKEL(Q) thus is $O(n^4)$. A lower bound of $\Omega(n^2\alpha(n)^2)$ has been shown in [4].

The construction time depends on the number of occurring events, which tends to stay linear for many inputs, as observed in our implementation. The detection of solid events is costly, though, due to the absence of a three-dimensional analogue of motorcycle graphs [8,12]. The direct method checks self-intersection of the offset polytope after each potential surface event, by intersecting $O(n)$ boundary triangles in $O(n^2)$ time. A theoretical speed-up to $O(n^{4/5+\varepsilon}k^{4/5+\varepsilon})$ is possible, provided the number k of triangle intersections is small [5].

The cells of SKEL(Q) are *monotone* in direction normal to their defining polytope facets. This can be shown by generalizing the 'roof' argument in [1] to one dimension higher. The monotonicity of skeleton cells may be useful in applications arising, for instance, in automatic meshing.

When not exclusively using Sph($\mathcal{P}$) but also other bisector graphs, offsets with tailor-made features can be generated. We have investigated this issue in a companion paper [3]. In particular, offset polytopes with a minimum or a maximum number of reflex edges are generated. There can be super-exponentially many (in n) different straight skeletons for a given polytope with n vertices. Finding skeletons efficiently that require a minimum number of *constructing events* (or even characterizing them in terms of vertex figure resolution) is of vital interest, as such skeletons have a smallest number of vertices.

The skeleton construction approach in Sections 2 and 3 works in principle in any dimension, though details get involved rapidly. As a possibly still tractable instance, a spherical skeleton in $\mathbb{R}^4$ can be defined on the basis of SKEL(Q), leading to a canonical straight skeleton for 4-dimensional polytopes.

Acknowledgements. We thank Günter Rote for interesting discussions.

References

1. Aichholzer, O., Alberts, D., Aurenhammer, F., Gärtner, B.: A novel type of skeleton for polygons. Journal of Universal Computer Science 1, 752–761 (1995)
2. Aurenhammer, F.: Weighted skeletons and fixed-share decomposition. Computational Geometry: Theory and Applications 40, 93–101 (2007)
3. Aurenhammer, F., Walzl, G.: Three-dimensional straight skeletons from bisector graphs. In: Proc. 5th International Conference on Analytic Number Theory and Spatial Tessellations, Kiev, Ukraine (to appear, 2013)
4. Barequet, G., Eppstein, D., Goodrich, M.T., Vaxman, A.: Straight skeletons of three-dimensional polyhedra. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 148–160. Springer, Heidelberg (2008)
5. De Berg, M., Guibas, L.J., Halperin, D.: Vertical decompositions for triangles in 3-space. Discrete & Computational Geometry 15, 35–61 (1996)
6. Demaine, E.D., Demaine, M.L., Lindy, J.F., Souvaine, D.L.: Hinged dissection of polypolyhedra. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 205–217. Springer, Heidelberg (2005)
7. Devadoss, S.L., O'Rourke, J.: Discrete and Computational Geometry. Princeton University Press (2011)
8. Eppstein, D., Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Discrete & Computational Geometry 22, 569–592 (1999)
9. Grünbaum, B.: Convex Polytopes. Interscience, New York (1967)
10. Martinez, J., Vigo, M., Pla-Garcia, N.: Skeleton computation of orthogonal polyhedra. Computer Graphics Forum 30, 1573–1582 (2011)
11. Siddiqi, K., Pizer, S.M.: Medial Representations. Mathematics, Algorithms, and Applications. Springer Series on Computational Imaging and Vision 37 (2008)
12. Vigneron, A., Yan, L.: A faster algorithm for computing motorcycle graphs. In: Proc. 29th Ann. ACM Symposium on Computational Geometry, pp. 17–26 (2013)

Pattern Matching with Non Overlapping Reversals - Approximation and On-line Algorithms

Amihood Amir[1,2,*] and Benny Porat[1,**]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
amir@cs.biu.ac.il, bennyporat@gmail.com
[2] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

Abstract. The *Sorting by Reversals* Problem is known to be $\mathcal{NP}$-hard. A simplification, *Sorting by Signed Reversals* is polynomially computable. Motivated by the Pattern Matching with Rearrangements model, we consider *Pattern Matching with Reversals*. Since this is a generalization of the Sorting by Reversals problem, it is clearly $\mathcal{NP}$-hard. We, therefore consider the simplification where reversals cannot overlap. Such a constrained version has been researched in the past for various metrics in the rearrangement model - the swap metric and the interchange metric. We show that the constrained problem can be solved in linear time. We then consider the *Approximate Pattern Matching with non-overlapping Reversals* problem, i.e. where mismatch errors are introduced. We show that the problem can be solved in quadratic time and space. Finally, we consider the on-line version of the problem. We introduce a novel signature for palindromes and show that it has a pleasing behavior, similar to the Karp-Rabin signature. It allows solving the *Pattern Matching with non-overlapping Reversals* problem on-line in linear time w.h.p.

1 Introduction

Consider a text $T = t_0 \cdots t_{n-1}$ and pattern $P = p_0 \cdots p_{m-1}$, both over an alphabet Σ. Traditional pattern matching regards T and P as *sequential* strings, provided and stored in sequence (e.g. from left to right). Therefore, implicit in the conventional approximate pattern matching is the assumption that there may indeed be errors in the **content** of the data, but the **order** of the data is inviolate. However, examples from text editing, bit torrent and video on demand, computer architecture, and computational biology, motivated a new pattern matching paradigm – *pattern matching with rearrangements*. For a survey on the rearrangement paradigm see [5].

In addition to the theoretical combinatorial motivation, the paradigm is also fueled by biological challenges. During the course of evolution areas of the

[*] Partly supported by NSF grant CCR-09-04581, ISF grant 347/09, and BSF grant 2008217.

[**] Partly supported by a Bar Ilan University President Fellowship. This work is part of Benny Porat's Ph.D. thesis.

genome may be shifted from one location to another. Considering the genome as a string over the alphabet of genes, these problems represent a situation where the difference between the original string and the resulting one is in the locations rather than contents of the different elements. Several works have considered specific versions of this biological setting, primarily focusing on the sorting problem (*sorting by reversals* [8,10], *sorting by transpositions* [6], and *sorting by block interchanges* [11]).

The *Min-SBR (Minimum Sorting by Reversals)* problem, gets a permutation $\pi = \pi[1], ..., \pi[n]$ of $\{1, ..., n\}$ as its input, and its task is to sort the permutation using the minimum number of possible *reversal* operations. A reversal operation reverses the order of a substring of the permutation, i.e.

$$rev_{(i,j)}(\pi[1], ..., \pi[i-1], \pi[\mathbf{i}], \pi[\mathbf{i+1}], ..., \pi[\mathbf{j-1}], \pi[\mathbf{j}], \pi[j+1], ..., \pi[n]) =$$

$$\pi[1], ..., \pi[i-1], \pi[\mathbf{j}], \pi[\mathbf{j-1}], ..., \pi[\mathbf{i+1}], \pi[\mathbf{i}], \pi[j+1], ..., \pi[n].$$

The Min-SBR problem is $\mathcal{NP}$-hard and has a long history. If the reversals are signed, there exist polynomial time algorithms [8,15]. For unsigned reversals there are efficient approximation algorithms [12,9].

This paper studies the reversals problem in a general alphabet setting, rather than as a sorting problem, i.e. symbols may occur in the strings multiple times. Consider a set A and let x and y be two m-tuples over A. We wish to convert x to y through a sequence of reversal operations. We call this problem the *General Alphabet Matching with Reversals Problem.*

Since the Sorting by Reversals problem is a special case of the General Alphabet Matching with Reversals problem, then the latter is also $\mathcal{NP}$-hard. We therefore consider a constrained version of this problem.

A very special case of the problem had been significantly studied. The *Pattern Matching with Swaps* problem (the *Swap Matching* problem, for short), defined by Muthukrishnan [20], requires finding all occurrences of a pattern of length m in a text of length n. The pattern is said to match the text at a given location i if adjacent pattern characters can be swapped, if necessary, so as to make the pattern identical to the substring of the text starting at location i. All the swaps are constrained to be disjoint, i.e., each character is involved in at most one swap.

Swap Matching is a specialized case of General Alphabet Matching by Reversals. It has two constraints:

Length: The length of a reversed substring is limited to 2.
Disjointness: All swaps are disjoint, i.e. each symbol can participate in at most one swap.

Both these constraints together tremendously simplified the problem. While the General Alphabet Matching by Reversals problem is $\mathcal{NP}$-hard, a series of papers [2,4,13] culminated in a $O(n \log m \log \sigma)$ time algorithm [3] for the Swap Matching problem, where n is the length of the text, m is the length of the pattern and σ is the minimum of the pattern length and the alphabet size. It should be noted that unlike the definition of the Sorting by Reversals problem or the General Alphabet Matching with Reversals problem, the Swap Matching problem "shifts" the pattern over a longer text, computing the swap match in every text location. The reversal problems we defined above consider two strings of the same length, thus no shift is done.

It is natural, then, to ask, what happens if only one of the above constraints exists. If only the length constraint is kept, the problem becomes similar to that of the time complexity of *Bubble-sort*. In this paper we consider the problem of loosening the length constraint but keeping the disjointness constraint.

Definition 1. *The* Reversal Matching Problem *is the General Alphabet Matching with Reversals problem with the disjointness constraint, i.e., each symbol can participate in at most one reversal.*

We say that string S reversal matches *string T if string T results from performing a sequence of disjoint reversal operations on string S.*

This natural constraint has been used not only in the Swap Matching problem but also in other contexts of the rearrangements model, for example in the Pattern Matching with Interchanges problem [1].

We summarize the differences between the above definition of the reversal matching problem and the traditional sorting by reversals problem (Min-SBR). The Min-SBR problem is a sorting problem (i.e. every symbols appears once in the string) whereas the reversal matching problem is a pattern matching problem, i.e. symbols can repeat. In the Min-SBR problem, the same symbol may be part of several reversal operations, whereas in the reversal matching problem if a symbol is part of a reversal operation, it cannot be moved again. Every permutation can be sorted by reversals, consequently the interesting question in the Min-SBR problem is finding the minimum number of reversals necessary for sorting. However, because of the disjointness constraint, even strings having the same alphabet and the same number of occurrences for each symbol, do not necessarily reversal match, thus the question that interests us is whether two strings reversal match. If they do, there is only one sequence of disjoint reversal operations that achieves the reversals match.

The contributions of this paper are theoretical in nature:

1. Considering the Sorting by Reversals problem in the context of the Pattern Matching with Rearrangements model.
2. A combinatorial analysis and understanding of the non-overlapping reversals phenomenon.
3. Formalizing and exploiting the intuitive relation between reversals and palindromes.
4. A novel fingerprint idea that enables on-line sorting by reversals, and on-line palindrome detection.

We next consider the *approximate reversal matching problem.* Given two strings, s_1 and s_2, does there exist a string s'_2 that reversal matches s_1, and such that the Hamming distance between s_1 and $'_2$ is minimized, or is smaller than a given k. We provide an algorithm that can compute the k-reversal distance of strings s_1 and s_2

of length n in time $O(n^2k)$ and linear space, and an algorithm that computes the k-reversal distance in time and space $O(n^2)$. We also show a trade-off algorithm whose time is $O(n^2 \log k)$ and whose space is $O(nk)$.

2 Reversal Matching up to Distance *k*

2.1 Exact Reversal Matching

We start with an algorithm for solving the Reversal Matching problem *without* mismatches.We first need some useful notation. Let $s = s[1], s[2], \cdots, s[n-1], s[n]$ be a string.
We denote its *reverse* $s[n], s[n-1], \cdots, s[2], s[1]$ by s^R.
Let $1 \leq i \leq j \leq n$. Denote by $s[i : j]$ the substring $s[i], s[i+1], ..., s[j-1], s[j]$.
Let $1 \leq j \leq n$. We denote the *length-j prefix of* s, $s[1 : j]$ by $Pre_j(s)$. We denote the *suffix of s starting at position j*, $s[j : n]$, by $Suf_j(s)$.

The reversal property below enables a surprisingly simple greedy linear time algorithm for the exact reversal matching.

Lemma 1. The Reversal Property: *Let s be string of length n and let $1 \leq j < n$. If $Pre_j(s) = Pre_j(s^R)^R$ then $Suf_{j+1}(s)$ and $Suf_{j+1}(s^R)$ reversal match.*

Proof: $s = s[1] \cdots s[n]$ and $s^R = s[n] \cdots s[1]$. $Pre_j(s) = s[1] \cdots s[j]$ and $Pre_j(s^R) = s[n]s[n-1] \cdots s[n-j+1]$. We are given that $Pre_j(s) = Pre_j(s^R)^R$, meaning $s[1] \cdots s[j] = s[n-j+1] \cdots s[n]$. The lemma's condition, then, implies that the first j symbols of s are equal to the last j symbols of s, i.e. $Pre_j(s) = Suf_{n-j+1}(s)$ and, consequently, $Pre_j(s^R) = Suf_{n-j+1}(s^R)$.

We get $Suf_{n-j+1}(s) = Pre_j(s) = Pre_j(s^R)^R = Suf_{n-j+1}(s^R)^R$.

Consider now two cases:

1. If $n > 2j$ then the first j symbols of s do not overlap the last j symbols. However, for every string t we have $t = (t^R)^R$ so, in particular, $s[j+1 : n-j] = (s^R[j+1 : n-j])^R$. $Suf_{j+1}(s)$ is $s[j+1..n-j]$ concatenated with $Suf_{n-j+1}(s)$. $Suf_{j+1}(s^R)$ is $s^R[j+1 : n-j]$ concatenated with $Suf_{n-j+1}(s^R)$. But $s[j+1 : n-j] = (s^R[j+1 : n-j])^R$ and $Suf_{n-j+1}(s) = Suf_{n-j+1}(s^R)^R$, thus $Suf_{j+1}(s)$ and $Suf_{j+1}(s^R)$ reversal match.

2. If $n \leq 2j$ then the first j symbols overlap with the last j symbols. Nevertheless, we have seen that they are equal. Let s_o be the substring where the first j symbols of s overlap with the last j symbols, and let $x = |s_o$. We have that $Suf_{j+1}(s)$ consists of $s_1 = s[x+1] \cdots s[j-x]$ followed by s_o. $Suf_{j+1}(s^R)$ is s_1^R concatenated with s_o^R, as can be seen in Figure 1. □

The reversal property immediately suggests a greedy algorithm for reversal matching.

Greedy Algorithm Outline: Given s_1 and s_2, go from left to right. Start from index $i = 1$.
Assuming we have shown that s_1 and s_2 reversal match until index i, find the shortest substring s' starting at $s_1[i]$ such that s'^R is the substring of s_2 starting

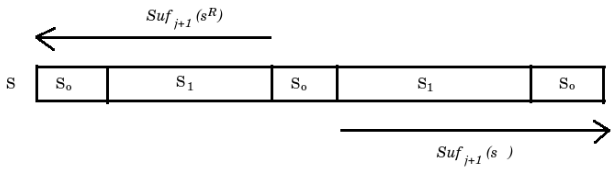


Fig. 1. Overlapping prefix and suffix of s

at $s_2[i]$. If the length of s' is j then the next index to check is $i + j$. If no such substring exists then s_1 does not reversal match s_2.

Algorithm Correctness: The reason this greedy strategy works is the following. Suppose there is a reversal matching where the chosen reversal at position i is not the shortest. Then at index i there is a longer substring s'' that starts at $s_1[i]$ such that s''^R is the substring of s_2 starting at $s_2[i]$. Because of the reversal property, reversing s' will still mean that the remaining suffix of s'' in s_1 reverse matches the remaining suffix of s''^R in s_2.

Algorithm Time: Finding s' can be done in time $|s'|$ by checking, for each prefix $Pre_i(s')$, $i = 1, ..., |s'|$ of s', whether $Pre_i(s')^R$ equals the corresponding prefix in s_2. This can be done in a standard way by LCA queries on a suffix tree of s_1 and s_2^R (e.g. [21,7]), or by LCP queries on a suffix array of s_1 concatenated to S_2^R (e.g. [16,18]). These queries take constant time following a linear-time preprocessing for fixed finite alphabets. There is a multiplicative $\log \sigma$ factor for general alphabets, where $\sigma = \min(n, |\Sigma|)$.

2.2 The Mismatch Case - Dynamic Programming

We now introduce mismatch errors into the problem. The philosophy is akin to the prevailing one in pattern matching. We assume that mismatch errors have been introduced to a string s_2. We seek the smallest number of such mismatches that, if fixed, will make s_2 reversal match s_1. We formally define this below, and introduce a strong tool for an efficient solution – the *palindrome distance.*

Definition 2. *Let s_1, s_2 be strings of length n. Denote by $Ham(s_1, s_2)$ the Hamming distance between s_1 and s_2, i.e. the number of mismatches between them. The* palindrome distance *between s_1 and s_2, denoted by $PD(s_1, s_2)$ is $Ham(s_1, s_2^R)$. Denote*

$$PDk(s_1, s_2) = \begin{cases} PD(s_1, s_2) & \text{if } PD(s_1, s_2) \le k, \\ \infty & \text{otherwise.} \end{cases}$$

The reversal distance *between s_1 and s_2, denoted by $RD(s_1, s_2)$ is the minimum number k such that there exists a string s_2', where $Ham(s_2, s_2') = k$ and where s_1 reversal matches s_2'.*

The k-reversal distance *between s_1 and s_2, denoted by $RDk(s_1, s_2)$ is*

$$RDk(s_1, s_2) = \begin{cases} RD(s_1, s_2) & \text{if } RD(s_1, s_2) \le k, \\ \infty & \text{otherwise.} \end{cases}$$

We are interested in efficiently finding the k-reversal distance. The problem with introducing mismatches is that the reversal property breaks down when dealing with mistakes. It is therefore necessary to actually check all possibilities of reversal matching the strings. This can be done using dynamic programming.

Dynamic Programming Algorithm Outline.
Let A be an array of length $n+1$. The cell $A[i]$ will hold the reversal distance of the two length-i prefixes of the strings, i.e. $RD(Pre_i(s_1), Pre_i(s_2))$. The array can be constructed as follows:

Initialize $A[0] \leftarrow 0$.

Assuming we have all the values of $A[\ell]$, $\ell = 0, ..., i-1$. Compute $A[i]$ in the following manner: For all $\ell = 0, ..., i-1$, calculate the palindrome distance between $s_1[\ell : i]$ and $s_2[\ell : i]$ plus the reversal distance between $Pre_\ell(s_1)$ and $Pre_\ell(s_2)$. Formally:

$$A[i] = \min_{0 \le \ell < i} \{A[\ell] + PDk(s_1[\ell : i], s_2[\ell : i])\}.$$

We will show below that, for a given i, we can calculate the palindrome distances up to k errors, $PDk(s_1[\ell : i], s_2[\ell : i])$ for all $0 \le \ell \le i$ in time $O(ik)$. The Palindrome Distance algorithm works in an on-line fashion, i.e. on the ℓ-th step the algorithm reads the ℓ-th characters, $s_1[\ell]$ and $s_2[\ell]$, and outputs the palindrome distance $PDk(Pre_\ell(s_1), Pre_\ell(s_2))$. Each such step takes $O(k)$ time. Thus, in order to calculate $A[i]$ we need to calculate $PDk(s_1[\ell : i], s_2[\ell : i])$ for all $0 \le \ell < i$.

Total Running Time: The time to calculate $A[i]$ is $O(ik)$, so the total time is $O(n^2k)$.

It remains to show how one can calculate $PDk(s_1[\ell : i], s_2[\ell : i])$ in time $O(k)$. This is done by employing the *kangaroo* idea of Galil and Giancarlo [14]. Galil and Giancarlo used the Landau and Vishkin idea [19] of using LCA's on a suffix tree in order to compute the k-Hamming distance of a pattern P at location i of text T in time $O(k)$. This is precisely what we need to do for $s_1[\ell : i]$ and $s_2[\ell : i]^R$. The same idea is used, but s_2^R needs to be part of the suffix tree.

3 Speeding up the Dynamic Programming Algorithm

In this subsection we speed up the dynamic programming algorithm. The more efficient time complexity is achieved due to the following lemma.

Lemma 2. *For any* $0 \le j < i < n$ *and* $\tau < \frac{i-j}{2}$*:*
$PD(s_1[j+\tau : i-\tau], s_2[j+\tau : i-\tau]) = PD(s_1[j : i], s_2[j : i]) - (PD(s_1[j : j+\tau], s_2[i-\tau : i]) + PD(s_1[i-\tau : i], s_2[j : j+\tau]))$

Proof: Consider Figure 2.

The following string of equalities can be seen from Figure 2:
$PD([s_1[j : i], s_2[j : i]) = Ham(s_1[j : i], s_2[j : i]^R) =$
$= Ham(s_{11}s_{1c}s_{12}, (s_{21}s_{2c}s_{22})^R) = Ham(s_{11}s_{1c}s_{12}, s_{22}^R s_{2c}^R s_{21}^R) =$

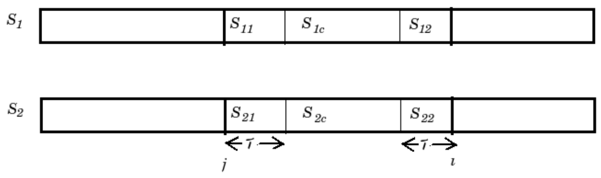


Fig. 2. Palindrome distance of $s_1[j : i]$ and $s_2[j : i]$

$= Ham(s_{11}, s_{22}^R) + Ham(s_{1c}, s_{2c}^R) + Ham(s_{12}, s_{21}^R) =$
$= PD(s_1[j : j+\tau], s_2[i-\tau : i]) + PD(s_1[j+\tau : i-\tau], s_2[j+\tau : i-\tau]) + PD(s_1[i-\tau : i], s_2[j : j+\tau])$. □

The lemma leads to a simple strategy for computing multiple palindrome distances fast. In essence, adding or deleting a pair of symbols on the two edges changes the palindrome distance depending on the equality or inequality of the added symbols.

Conclusion 1. *Given $s_1[j : i]$, $s_2[j : i]$, $\tau < \frac{i-j}{2}$ and $PD(s_1[j : i], s_2[j : i])$, one can calculate $PD(s_1[j + \mu : i - \mu], s_2[j + \mu : i - \mu])$ for all $\mu < \tau$ in $O(\tau)$ time.*

Proof: Start from $\mu = 0$. We are given $PD(s_1[j + 0 : i - 0], s_2[j + 0 : i - 0]) = PD(s_1[j : i], s_2[j : i])$. Assume we have already calculated $PD(s_1[j + \mu : i - \mu], s_2[j + \mu : i - \mu]) = pd_\mu$. Then $PD(s_1[j + \mu + 1 : i - (\mu + 1)], s_2[j + \mu + 1 : i - (\mu - 1)])$ gets the following value:

$$\begin{cases} pd_\mu & \text{if} \quad s_1[j+\mu+1] = s_2[i-(\mu+1)]) \text{ and } s_1[i-(\mu+1)] = s_2[j+\mu+1], \\ pd_\mu + 1 & \text{if} \quad s_1[j+\mu+1] = s_2[i-(\mu+1)]) \text{ and } s_1[i-(\mu+1)] \neq s_2[j+\mu+1], \\ pd_\mu + 1 & \text{if} \quad s_1[j+\mu+1] \neq s_2[i-(\mu+1)]) \text{ and } s_1[i-(\mu+1)] = s_2[j+\mu+1], \\ pd_\mu + 2 & \text{if} \quad s_1[j+\mu+1] \neq s_2[i-(\mu+1)]) \text{ and } s_1[i-(\mu+1)] \neq s_2[j+\mu+1]. \end{cases}$$ □

Conclusion 2. *Given $s_1[j : i]$, $s_2[j : i]$, one can calculate $PD(s_1[j + \mu : i - \mu], s_2[j + \mu : i - \mu])$ for all $\mu \leq \lceil \frac{i-j}{2} \rceil$ in $O(i - j)$ time.*

Proof: A distinction needs to be made between the case where the length $i-j+1$ of $s[j : i]$ is odd and where the length is even. If it is odd, then the element $s_1[j + \frac{i-j}{2}]$ is exactly in the middle of $s_1[j : i]$. Call it the *center element of* $s_1[j : i]$. Similarly, $s_2[j + \frac{i-j}{2}]$ is the center element of $s_2[j : i]$. We compute the palindrome distance from the center out. Start with $\mu = \frac{i-j}{2}$. $PD(s_1[j + \mu : i - \mu], s_2[j + \mu : i - \mu]) = PD(s_1[j + \frac{i-j}{2} : j + \frac{i-j}{2}], s_2[j + \frac{i-j}{2} : j + \frac{i-j}{2}])$. Its value is:

$$\begin{cases} 0 & \text{if} \quad s_1[j + \frac{i-j}{2}] = s_2[j + \frac{i-j}{2}], \\ 1 & \text{otherwise.} \end{cases}$$

Assume now, that we have computed all values of PD up to $PD(s_1[j + \mu : i - \mu], s_2[j + \mu : i - \mu]) = pd_\mu$. Compute the value of $PD(s_1[j + \mu - 1 : i - \mu + 1], s_2[j + \mu - 1 : i - \mu + 1])$ as follows:

$$\begin{cases} pd_\mu & \text{if} \quad s_1[j+\mu-1] = s_2[i-\mu+1] \text{ and } s_1[i-\mu+1] = s_2[j+\mu-1], \\ pd_\mu + 1 & \text{if} \quad s_1[j+\mu-1] = s_2[i-\mu+1] \text{ and } s_1[i-\mu+1] \neq s_2[j+\mu-1], \\ pd_\mu + 1 & \text{if} \quad s_1[j+\mu-1] \neq s_2[i-\mu+1] \text{ and } s_1[i-\mu+1] = s_2[j+\mu-1], \\ pd_\mu + 2 & \text{if} \quad s_1[j+\mu-1] \neq s_2[i-\mu+1] \text{ and } s_1[i-\mu+1] \neq s_2[j+\mu-1]. \end{cases}$$

If $|s[j:i]| = i-j+1$ is even then the algorithm is the same, but the center lies between two indices $\sigma = \lfloor \frac{i-j}{2}$ and $\sigma+1$. We call $s[\sigma]$ the *center element* of s.

Initialize PD for a nonexistent $\mu = \sigma+1$, with $PD(s_1[j+\mu : i-\mu], s_2[j+\mu : i-\mu]) = 0$. Subsequently, for μ going down from σ to 0 we compute the value of $PD(s_1[j+\mu-1 : i-\mu+1], s_2[j+\mu-1 : i-\mu+1])$ in the same manner as the odd-length case. □

Conclusion 3. *The palindrome distances $PD(s_1[i:j], s_2[i:j])$, $1 \leq i \leq j \leq m$ can be computed in time $O(n^2)$.*

Proof: There are $O(n^2)$ palindrome distances to calculate. By Conclusion 2, one can compute in time $O(n)$ the palindrome distances of all odd-length substrings whose center element is in index i. These are the distances:

$$\{PD(s_1[i-\tau : i+\tau], s_2[i-\tau : i+\tau]) \mid \tau = 0, \ldots \min\ (n-i, i)\}.$$

For even-length substrings, where the center element is in index i, one can similarly compute in time $O(n)$, all the values:

$$\{PD(s_1[i-\tau : i+1+\tau], s_2[i-\tau : i+1+\tau]) \mid \tau = 0, \ldots \min\ (n-i-1, i)\}.$$

The union of all above substrings, for $i = 0, \ldots, n$ for odd-length strings, and $i = 0, \ldots, n-1$ for even-length strings, is all relevant substrings, and can thus be computed in time $O(n^2)$. □

Conclusion 3 allows us to precompute all palindrome distances in time $O(n^2)$, and use these precomputed values for the dynamic programming algorithm of Section 2.2 to get a total algorithm time of $O(n^2)$. However, this algorithm requires $\Theta(n^2)$ space.

At this point we have an algorithm that can compute the k-reversal distance of strings s_1 and s_2 of length n in time $O(n^2 k)$ and linear space, and an algorithm that computes the k-reversal distance in time and space $O(n^2)$.

We show below a trade-off algorithm whose time is $O(n^2 \log k)$ and whose space is $O(nk)$.

Trade-Off Algorithm Outline.
Compute all palindrome distances of the $O(n^2)$ substrings as in Conclusion 3. However, rather than storing them in an $n \times n$ table, store them in an $n \times k$ table. For each substring centered at index i, write down the list of τ's where the palindrome distances are incremented.

Now, in the dynamic programming algorithm, when $PDk(s_1[\ell : i], s_2[\ell : 1])$ is required, simply compute the center and the τ and do a binary search on the $n \times k$ table to find the appropriate k-palindrome distance. This computation can be easily done in constant time. This leads to the following conclusion.

Conclusion 4. *The k-reversal distance of strings s_1 and s_2 of length n can be computed in time $O(n^2 \log k)$ and space $O(nk)$.*

While the dynamic programming scheme goes from left to right and is on-line, the efficient palindrome distance computations make use of mechanisms that are not on-line. Although there exist algorithms that compute suffix trees on-line, it is not known how to do both suffix trees and LCA on-line. To this end we develop an alternate algorithm to efficiently compute the palindrome distance. At the heart of this algorithm is a novel fingerprint scheme that detects palindromes with high probability. This algorithm is also interesting because it is essentially the Karp-Rabin algorithm [17] but with a different fingerprint formula. It demonstrates nicely how different fingerprints can solve different pattern matching algorithms.

4 A Fingerprint-Based Algorithm for the Reverse Matching Problem

4.1 Reversals and Palindrome Recognition

This subsection shows the direct relation between reversal matching and recognizing palindromes.

Definition 3. *Let $s_1 = s_1[1], s_1[2], \cdots, s_1[n-1], s_1[n]$ and $s_2 = s_2[1], s_2[2], \cdots, s_2[n-1], s_2[n]$ be two length-n strings. The* interleaved string of s_1 and s_2 *is the string*

$$Int(s_1, s_2) = s_1[1], s_2[1], s_1[2], s_2[2], \cdots, s_1[n-1], s_2[n-1], s_1[n], s_2[n].$$

Example: Let $s_1 = ABCD$ and $s_2 = DCBA$.
$Int(s_1, s_2) = ADBCCBDA$.

The *interleave lemma* below makes the connection between reversal matching and palindromes.

Lemma 3. The Interleave Lemma: *Let s_1 and s_2 be two length-n strings. Then $s_2 = s_1^R$ iff $Int(s_1, s_2)$ is a palindrome.*

We are now ready to define the key property enabling our algorithm's on-line efficiency.

Definition 4. Palindrome fingerprint*: Given a string $s = s[1], s[2], ...s[m]$ over $\mathbb{N}$, and some random number $r \in F_p$, we define the* palindrome fingerprint, *or for brevity the* fingerprint *of s to be: $\phi(s) = r^1 s[1] + r^2 s[2] + ...r^m s[m]$. We define the* reversal fingerprint *of s, to be: $\phi^R(s) = r^{-1}s[1] + r^{-2}s[2] + ...r^{-m}s[m]$*

Lemma 4. *Given a string s, $\phi(s) = r^{m+1}\phi^R(s)$ iff s is a palindrome ($s = s^R$) w.h.p.*

4.2 Exact Linear-time Online Reversal Matching

The greedy exact reversal matching algorithm presented in Section 2.1 takes time $O(n^2)$ as an on-line algorithm. We added the Suffix tree and LCA mechanism to reduce it to linear time, but these mechanisms are not on-line. They can be replaced by the palindrome fingerprint and the interleave lemma. Recall that the greedy strategy works as follows. Suppose there is a reversal matching ending at position i. We seek the shortest substring s' that starts at $s_1[i]$ such that s'^R is the substring of s_2 starting at $s_2[i]$. Assume the length of s' is ℓ. The situation is that $s_1[i:j]^R \neq s_2[i:j],\ j = i, ..., \ell-1$, but $s_1[i:\ell]^R = s_2[i,\ell]$.

By the interleave lemma $s_1[i:j]$ interleaved with $s_2[i:j]$ is not a palindrome, for $j = i, ..., \ell-1$, but $s_1[i:\ell]$ interleaved with $s_2[i:\ell]$ is a palindrome. The crucial point is that it takes $O(1)$ time to extend the fingerprint and the reversal fingerprint from $s_1[i:j]$ interleaved with $s_2[i:j]$ to $s_1[i:j+1]$ interleaved with $s_2[i:j+1]$. We show this for the fingerprint, the reversal fingerprint is similar. The fingerprint of $s_1[i:j]$ interleaved with $s_2[i:j]$ is

$$\phi_j = r^1 s_1[i] + r^2 s_2[i] + r^3 s_1[i+1] + r^4 s_2[i+1] + ... + r^{2(j-i)+1} s_1[j] + r^{2(j-i+1)} s_2[j].$$

The fingerprint ϕ_{j+1} of $s_1[i:j+1]$ interleaved with $s_2[i:j+1]$ is $r^1 s_1[i] + r^2 s_2[i] + r^3 s_1[i+1] + r^4 s_2[i+1] + ... + r^{2(j-i)+1} s_1[j] + r^{2(j-i+1)} s_2[j] + r^{2(j+1-i)+1} s_1[j+1] + r^{2(j+1-i+1)} s_2[j+1]$ I.e. $\phi_{j+1} = \phi_j + r^{2(j+1-i)+1} s_1[j+1] + r^{2(j+1-i+1)} s_2[j+1]$.

By Lemma 4, it also takes constant time to check whether each $s_1[i:j]$ interleaved with $s_2[i:j]$ is a palindrome. We conclude:

Conclusion 5. *The exact reversal matching of strings s_1 and s_2 of length n can be computed on-line in time $O(n)$ w.h.p.*

References

1. Amir, A., Aumann, Y., Benson, G., Levy, A., Lipsky, O., Porat, E., Skiena, S., Vishne, U.: Pattern matching with address errors: rearrangement distances. J. Comp. Syst. Sci. 75(6), 359–370 (1995)
2. Amir, A., Aumann, Y., Landau, G., Lewenstein, M., Lewenstein, N.: Pattern matching with swaps. Journal of Algorithms 37, 247–266 (2000) (Preliminary version appeared at FOCS 1997)
3. Amir, A., Cole, R., Hariharan, R., Lewenstein, M., Porat, E.: Overlap matching. Information and Computation 181(1), 57–74 (2003)
4. Amir, A., Landau, G.M., Lewenstein, M., Lewenstein, N.: Efficient special cases of pattern matching with swaps. Information Processing Letters 68(3), 125–132 (1998)
5. Amir, A., Levy, A.: String rearrangement metrics: A survey. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) Ukkonen Festschrift 2010. LNCS, vol. 6060, pp. 1–33. Springer, Heidelberg (2010)
6. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. on Discrete Mathematics 11, 221–240 (1998)

7. Berkman, O., Vishkin, U.: Finding level-ancestors in trees. Journal of Computer and System Sciences 48(2), 214–229 (1994)
8. Berman, P., Hannenhalli, S.: Fast sorting by reversal. In: Hirschberg, D.S., Meyers, G. (eds.) CPM 1996. LNCS, vol. 1075, pp. 168–185. Springer, Heidelberg (1996)
9. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-approximation algorithm for sorting by reversals. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 200–210. Springer, Heidelberg (2002)
10. Carpara, A.: Sorting by reversals is difficult. In: Proc. 1st Annual International Conference on Research in Computational Biology (RECOMB), pp. 75–83. ACM Press (1997)
11. Christie, D.A.: Sorting by block-interchanges. Information Processing Letters 60, 165–169 (1996)
12. Christie, D.A.: A 3/2-approximation algorithm for sorting by reversals. In: Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 244–252 (1998)
13. Cole, R., Hariharan, R.: Randomized swap matching in o(m logm log$|\sigma|$) time. Technical Report TR1999-789, New York University, Courant Institute (September 1999)
14. Galil, Z., Giancarlo, R.: Improved string matching with k mismatches. SIGACT News 17(4), 52–54 (1986)
15. Kaplan, H., Shamir, R., Tarjan, R.E.: A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comp. 29(3), 880–892 (1999)
16. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 943–955. Springer, Heidelberg (2003)
17. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM Journal of Res. and Dev., 249–260 (1987)
18. Kasai, T., Lee, G.H., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
19. Landau, G.M., Vishkin, U.: Efficient string matching in the presence of errors. In: Proc. 26th IEEE FOCS, pp. 126–126 (1985)
20. Muthukrishnan, S.: New results and open problems related to non-standard stringology. In: Galil, Z., Ukkonen, E. (eds.) CPM 1995. LNCS, vol. 937, pp. 298–317. Springer, Heidelberg (1995)
21. Ukkonen, E.: On-line construction of suffix trees. Algorithmica 14, 249–260 (1995)

Single and Multiple Consecutive Permutation Motif Search

Djamal Belazzougui[1], Adeline Pierrot[2,⋆], Mathieu Raffinot[3], and Stéphane Vialette[4]

[1] Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland
[2] Institute of Discrete Mathematics and Geometry, TU Wien, Wiedner Hauptstrasse 8-10, 1040 Wien, Austria
[3] LIAFA, Univ. Paris Diderot - Paris 7, 75205 Paris Cedex 13, France
[4] LIGM CNRS UMR 8049, Université Paris-Est, France
vialette@univ-mlv.fr

Abstract. Let t be a permutation (that shall play the role of the *text*) on $[n]$ and a motif p be a sequence of m distinct integer(s) of $[n]$, $m \leq n$. The motif p occurs in t in position i if and only if $p_1 \dots p_m$ is order-isomorphic to $t_i \dots t_{i+m-1}$, that is, for all $1 \leq k < \ell \leq m$, $p_k > p_\ell$ if and only if $t_{i+k-1} > t_{i+\ell-1}$. Searching for a motif p in a text t consists in identifying all occurrences of p in t. We first present a forward automaton which allows us to search for p in t in $O(m^2 \log\log m + n)$ time. We then introduce a Morris-Pratt automaton representation of the forward automaton which allows us to reduce this complexity to $O(m \log\log m + n)$ at the price of an additional amortized constant term. The latter automaton occupies $O(m)$ space. We then extend the problem to search for a set of motifs and exhibit a specific Aho-Corasick like algorithm. Next we present a sub-linear average case search algorithm running in $O\left(\frac{m \log m}{\log\log m} + \frac{n \log m}{m \log\log m}\right)$ time, that we eventually prove to be optimal on average.

1 Introduction

Two sequences of distinct integers are *order-isomorphic* if the permutations required to sort them are the same. A sequence p is said to be a *motif* (or *occurs*) within a sequence t if t has a *subsequence* that is order-isomorphic to p. Motif involvement in *permutations* and sequences has now become a very active area of research [9]. However, only few results on the complexity of finding motifs in permutations and sequences are known. It appears to be a difficult problem to decide given two permutations π and σ whether σ occurs in π, and in this generality the problem is NP-complete [7]. Let $[n]$ be the set of all integers from 1 to n and let S_n be the set of all permutations on $[n]$. For $\sigma \in S_m$ and $\pi \in S_n$, the $O(n^m)$ time brute-force algorithm was improved to $O(n^{0.47m+o(m)})$ time in

⋆ Most of the work was done when the author was in LIAFA.

[2]. There are several ways in which this notion of permutation motifs may be generalized, and we focus here on *consecutive motifs* (*i.e.* the match is required to consist of contiguous elements) [9]. A sequence p is said to be a *consecutive motif* or *consecutively occurs* within a sequence t if t has a *substring* that is order-isomorphic to p. Searching for a motif p in a text t consists in identifying all occurrences of p in t. Recently, using a modification of the classical Knuth-Morris-Pratt string matching algorithm, a $O(n + m \log m)$ time algorithm has been proposed for checking if a given sequence t of length n contains a substring which is order-isomorphic to a given motif p of length m [10]. The time complexity reduces to $O(n + m)$ time under the assumption that the symbols of the motif can be sorted in $O(m)$ time.

Let t be a permutation of length n and p be a sequence of $m \leq n$ distinct integers in $[n]$. First we present a forward automaton which allows us to search for p in t in $O(m^2 \log\log m + n)$ time. Next, we introduce a Morris-Pratt automaton representation [11] of the forward automaton which allows us to reduce this complexity to $O(m \log\log m + n)$ at the price of an additional amortized constant term for each symbol of the text. The latter automaton occupies $O(m)$ space while the former occupies $O(m^2)$ space. We then extend the problem to search for a set of motifs and exhibit a specific Aho-Corasick like algorithm. Finally we present a sub-linear average case search algorithm running in $O\left(\frac{m \log m}{\log\log m} + \frac{n \log m}{m \log\log m}\right)$ time that we eventually prove to be optimal on average. Both lower and upper bounds assume all text permutations to be equiprobable and all integer values in a motif to be distinct.

Let us define some notations. Let $\Sigma_n = [n]$. Abusing notations, we consider in this paper permutations of S_n as strings without symbol repetition, and we denote by Σ_n^* the set of all strings without symbol repetition (including the empty string), where each symbol is an integer in $[n]$. A *prefix* (resp. *suffix*, *factor*) u of p is a string such that $p = uw, w \in \Sigma_n^*$ (resp. $p = wu, w \in \Sigma_n^*$, $p = wuz, w, z \in \Sigma_n^*$). We also denote by $|w|$ the number of integer(s) in a string $w, w \in \Sigma_n^*$. We eventually denote by p^r the reverse of p, that is, the string formed by the symbols of p read in the reverse order. We denote by $p^{\equiv}$ the set of words of Σ_n^* which are order-isomorphic to p.

The following property is useful for designing automaton transitions.

Property 1. Let $p = p_1 \dots p_m \in \Sigma_n^*$ and $w = w_1 \dots w_\ell \in \Sigma_n^*$, $\ell < m$, such that w is order-isomorphic to $p_1 \dots p_\ell$, and let $\alpha \in [n]$ s.t. $w\alpha \in \Sigma_n^*$. Testing if $w\alpha$ is order-isomorphic to $p_1 \dots p_\ell p_{\ell+1}$ can be performed in constant time using only a pair of integers.

Proof. The pair of integers (x_1, x_2) is determined as follows: $x_1 \leq \ell$ is the position of the largest number p_{x_1} in $p_1..p_\ell$ which is smaller than $p_{\ell+1}$, if any. Otherwise, we fix x_1 arbitrarily to $-\infty$. Let $x_2 \leq \ell$ be the position of the smallest integer p_{x_2} in $p_1..p_\ell$ which is larger than $p_{\ell+1}$, if any. Otherwise, we fix x_2 to $+\infty$. Now, it suffices to test if $w_{x_1} < \alpha < w_{x_2}$ to check whether $w\alpha$ is order-isomorphic to $p_1 \dots p_{\ell+1}$ □

We define a function $\text{rep}(p = p_1 \dots p_m, j)$ which returns a pair of integers (x_1, x_2) that represents the pair defined in Property 1 for the prefix of length j of the motif p.

2 Tools

Before proceeding, we first describe some useful data structures we shall use as basic subroutines of our algorithms. The problem called *predecessor search problem* is defined as follows: given a set $S = \{x_1, x_2, \dots, x_n\} \subset [u]$ (u is called the size of the universe), we support the following query: given an integer y return its predecessor in the set S, namely the only element x_i such that $x_i \leq y < x_{i+1}$ [1]. In addition, in the dynamic case, we also support updates: add or remove an element from the set S. The standard data structures to solve the predecessor search are the balanced binary search trees [1,5]. They use linear space and support queries and updates in worst-case $O(\log n)$ time. However, there exist better data structures that take advantage of the structure of the integers to get better query and update time. Specifically, the Van-Emde-Boas tree [13] supports queries and updates in (worst-case) time $O(\log\log u)$ using $O(u)$ space. Using randomization, the y-fast trie achieves $O(n)$ space with queries supported in time $O(\log\log u)$ and updates supported in randomized $O(\log\log u)$ time. The problem has received series of improvements which culminated with Andersson and Thorup's result [4]. They achieve $O(n)$ space with queries and updates supported in $O(\min(\log\log u, \sqrt{\frac{\log n}{\log\log n}}))$ (the update time is still randomized).

A special case occurs when space u is available and the set of keys S is known to be smaller than $\log^c u$ for some constant c. In this case all operations are supported in worst-case constant time using the atomic-heap [14].

3 Forward Search Automaton

The problem we consider is to search for a motif p in a permutation t without preprocessing the text itself. By analogy to the simpler case of the direct search of a word p in text t, we build an automaton that recognizes $(\Sigma_n^*) \cdot p^{\equiv}$.

We formally define our forward search automaton $\mathcal{FD}(p)$ built on $p = p_1 \dots p_m$ as follows (see Figure 1 for an example): *(i)* $m + 1$ states corresponding to each prefix (including the empty prefix) of p, state 0 is initial, state m is terminal;
(ii) m forward transitions from state j to $j + 1$ labeled by $\text{rep}(p, j + 1)$;
(iii) some backward transitions $\delta(x, [i, j])$, where x numbers a state, $0 \leq x \leq m$, $i \in \{1, \dots, x\} \cup \{-\infty\}$, $j \in \{1, \dots, x\} \cup \{+\infty\}$, defined the following way: $\delta(x, [i, j]) = q$ if and only if for all $p_i < \alpha < p_j$ (resp. $\alpha < p_j$ if $i = -\infty$, $p_i < \alpha$ if $j = +\infty$), the longest prefix of p that is order-isomorphic to a suffix of $p_1 \dots p_x\alpha$ is $p_1 \dots p_q$. We also impose some constraints on outgoing transitions:

[1] By convention, if all the elements of S are larger than y, then return $-\infty$ and if no elements is larger than y then return x_n.

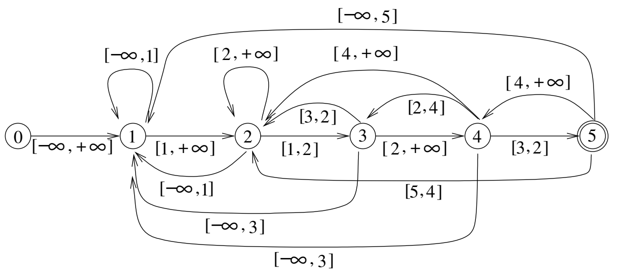


Fig. 1. Forward automaton built on $p = (4, 12, 6, 16, 10)$. State 0 is initial and state 5 is terminal.

Let x be the state corresponding to the prefix $p_1 \dots p_x$. Let us sort all $p_i, 1 \leq i \leq x$ and consider the resulting order $p_{i_0} = -\infty < p_{i_1} < \dots < p_{i_k} < +\infty = p_{i_{k+1}}$. We build one outgoing transition for each interval $[p_{i_j}, p_{i_{j+1}}]$, except if $p_{i_{j+1}} = p_{i_j} + 1$. Also we merge transitions that start in the same state and end if the same state whenever they are labeled by contiguous intervals.

It is obvious that the resulting automaton recognizes a given motif in a permutation by reading one by one each integer and choosing the appropriate transition. The main result on the forward automaton is the following.

Lemma 1. *Searching for a consecutive motif $p = p_1 \dots p_m$ in a permutation $t = t_1 \dots t_n$ using the forward automaton $\mathcal{FD}(p)$ built on p takes $O(n)$ time.*

We can build the forward automaton in $O(m^2 \log \log m)$ time. However, we defer the proof of this construction for the following reason. This $O(m^2 \log \log m)$ complexity might be too large for long motifs. Nevertheless, we show below that we can compute in a first step a type of Morris-Pratt coding of this automaton which can either (a) be directly used for the search for the motif in the text and will preserve the linear time complexity at the cost of an amortized constant term (we take more time for each text symbol), or (b) be developed to build the whole forward automaton structure.

Therefore we present and build a new automaton $\mathcal{MP}$ that is a Morris-Pratt representation of the forward automaton. The idea is to avoid building all backward transitions by only considering a special backward single transition from each state $x, x > 0$ named *failure* transition. We formally define our automaton $\mathcal{MP}(p)$ built on $p = p_1 \dots p_m$ the following way (see Figure 2 for an example):
(i) $m + 1$ states corresponding to each prefix (including the empty prefix) of p, state 0 is initial, state m is terminal;
(ii) m forward transitions from state j to $j + 1$ labeled by $\text{rep}(p, j + 1)$;
(iii) m failure (non labeled) transitions which connect a state $j > 0$ to a state $k < j$ if and only if $p_1 \dots p_k$ is the longest order-isomorphic border of $p_1 \dots p_j$:

Definition 1. *Let $p \in \Sigma_n^*$. A* border *of p is a word $w \in \Sigma_n^*, |w| < |p|$ that is order-isomorphic to a suffix of p but also order-isomorphic to a prefix of p.*

Reading a text t through the $\mathcal{MP}$ representation of the forward automaton is performed the following way. Let us assume we reached state $x < m$ and we

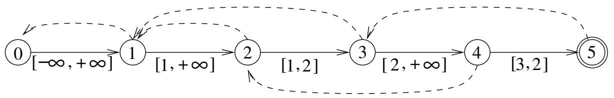


Fig. 2. $\mathcal{MP}$ automaton built on $p = (4, 12, 6, 16, 10)$. State 0 is initial and state 5 is terminal. Backward transitions are failure transitions.

read a symbol t_i at position i of the text. Let $[k, \ell] = \text{rep}(p, x+1)$. If $t_i \in [t_{i-x-1+k}, t_{i-x-1+\ell}]$ we follow the forward transition and the new current state is $x+1$. Otherwise, we *fail* reading t_i from x and we retry from state $q = \text{fail}(x)$ and so-on until (a) either q is undefined, in which case we start again from state 0, (b) or a forward transition from q to $q+1$ works, in which case the next current state is $q+1$.

Lemma 2. *Searching for a motif p in a text $t_1 \ldots t_n$ using the Morris-Pratt representation $\mathcal{MP}(p)$ of the forward automaton built on p takes $O(n)$ time.*

In order to prove Lemma 2 we need to focus on the classical notion of border that we have extended to our framework in Definition 1.

The construction of the forward automaton relies on the maximal border of each prefix that is followed by an appropriate integer in the motif. The Morris-Pratt approach is based on the following property:

Property 2. A border of a border is a border.

This property allows us to replace the direct transition of the forward algorithm by a search along the borders, from the longest to the smallest, to identify the longest one that is followed by the appropriate integer. We state now that we can build the Morris-Pratt representation of the forward automaton efficiently.

Lemma 3. *Building a Morris-Pratt representation of the forward automaton on a consecutive motif $p = p_1 \ldots p_m$ can be performed in (worst-case) $O(m \log\log m)$ time.*

Lemma 2 and 3 allow us to state the main theorem of this section.

Theorem 1. *Searching for a consecutive motif $p = p_1 \ldots p_m$ in a permutation $t = t_1 \ldots t_n$ can be done in $O(m \log\log m + n)$ time.*

The Morris-Pratt representation of the forward automaton permits to search directly in the text at the price of larger amortized complexity (considering the constant hidden by the O notation) than that required by searching with the forward automaton directly. If the real time cost of the search phase is an issue, the forward automaton can be built from its Morris-Pratt representation.

Property 3. Building the forward automaton of a consecutive motif $p = p_1 \ldots p_m$ can be performed in $O(m^2 \log\log m)$ time.

An interesting point is that the construction of the forward automaton from its Morris-Pratt representation can also be performed in a lazy way, that is, when reading the text. The missing transitions are then built *on the fly* when needed.

4 Multiple Worst Case Linear Motif Searching

We can extend the previous problem defined for a single motif to a set of motifs S. We denote by d the number of motifs, by m the total length of the motifs and by r the length of the longest motif. For this problem we adapt the Aho-Corasick automaton [3] (or $\mathcal{AC}$ automaton for short). We first recall the classical construction of the $\mathcal{AC}$ automaton (for regular motifs). The $\mathcal{AC}$ automaton is a generalization of the $\mathcal{MP}$ automaton to a set of multiple motifs. We denote by P the set of prefixes of strings in S. In order to simplify the description we will assume that the set of motifs S is prefix-free. That is, we will assume that no motif is prefix of another. Extending the algorithm to the case where S is non-prefix free, should not pose any particular issue. The states of the $\mathcal{AC}$ automaton are defined in the same way as in the $\mathcal{MP}$ automaton. Each state t in the $\mathcal{AC}$ automaton corresponds uniquely to a string $q \in P$. The forward transitions are defined as follows: there exists a forward transition connecting state s corresponding to a prefix q to each state corresponding to an element $qc \in P$ (where c is a single symbol). Thus this definition of the forward transitions matches essentially the definition of the forward transitions in the $\mathcal{MP}$ automaton. The failure transitions are defined as follows: the failure transition from the state s corresponding to a prefix q goes to the state s' corresponding to the longest string q' such that $q' \in P$, q' is a suffix of q and $q' \neq q$. The matching using the $\mathcal{AC}$ automaton is done in the same way as in the $\mathcal{MP}$ automaton using the forward and failure transitions.

Our Extension of the $\mathcal{AC}$ Automaton. We could use exactly the same algorithm as the one used previously for our variant of the $\mathcal{MP}$ automaton with few differences. We describe our modification to the $\mathcal{AC}$ automaton to adapt it to the case of consecutive permutation matching (a similar result which has been independently discovered is described in [8]). An important observation is that we could have two or more elements of P that are both of the same length and order-isomorphic. Those two elements should have a single corresponding state in the $\mathcal{AC}$ automaton. Thus, if two or more elements of P are order-isomorphic we keep only one of them. For the forward transitions, we can associate a pair of positions (x_1, x_2) to each forward transition. Then we can check which transition is the right one from a state corresponding to a string q by checking the condition $t_{i-|q|-1+x_1} < t_i < t_{i-|q|-1+x_2}$ for every pair (x_1, x_2) and take the corresponding transition. As any state can have up to d outgoing transitions, the time taken to choose the transition would grow to $O(d)$. We reduce the time to $O(\log d)$ by organizing the forward transitions outgoing from the same state into a balanced binary search tree. That is we put at the root of the balanced binary search tree the pair of positions (x_1, x_2), where x_1 is the median of all transition pairs (sorting the pairs by p_{x_1} values), and then on the left (resp. right) subtree all transitions whose corresponding pairs (x_1 component) point to (resp. larger) smaller values in the motif.

Altenatively we can use a different approach based on a dynamic balanced binary search tree (or more sophisticated dynamic predecessor data structure).

With the use of a binary search tree, we can achieve $O(\log r)$ time to decide which transition to take. More precisely, each time we read t_i we insert the pair (t_i, i) into the binary search tree. The insertion uses the number t_i as the key. Now suppose that we only pass through forward transitions. Then a transition at step i is uniquely determined by: (1) the current state s corresponding to an element $q \in P$; (2) the position of the predecessor of t_i among $t_{i-|q|} \dots t_{i-1}$.

To determine the predecessor of t_i among $t_{i-|q|} \dots t_{i-1}$, the dynamic binary search tree should contain precisely the $|q|$ pairs corresponding to $t_{i-|q|} \dots t_{i-1}$.

In order to maintain the dynamic binary search tree we must do the following actions while passing through a failure or a forward transition: (1) whenever we pass through a forward transition at a step i we insert the pair (t_i, i); (2) whenever we pass through a failure transition from a state corresponding to a prefix q_1 to a state corresponding to a prefix q_2, then we should remove from the binary tree all the pairs corresponding to the symbols $t_{i-|q_1|} \dots t_{i-|q_2|}$.

It should be noted that each removal or insertion of a pair into the binary search tree takes $O(\log r)$ time. The upper bound $O(\log r)$ comes from the fact that we never insert more than r elements in the binary search tree. Since in overall we are doing $O(n)$ insertions or removals, the amortized time should simplify to $O(n \log r)$. Finally if we replace binary search tree with a more efficient predecessor data structure, we will be able to achieve randomized time $O(n \cdot \tau)$ where $\tau = \min(\log\log n, \sqrt{\frac{\log r}{\log\log r}})$ is the time needed to do an operation on the predecessor data structure (see Section 2 for details). We use the linear space version of the predecessor data structure which guarantees only randomized performance but uses $O(r) \leq O(m)$ additional space only. We thus have the following theorem :

Theorem 2. *Searching in a text of size n for a set of d consecutive motifs whose $\mathcal{AC}$ automaton has been built and where the longest motif is of length r can be done in randomized $O(n \cdot \tau)$ time, where $\tau = \min(\log\log n, \sqrt{\frac{\log r}{\log\log r}}, \log d)$.*

Preprocessing. We now show that the preprocessing phase can be done in worst-case $O(m \log\log r)$ time. As before our starting point will be to sort all the motifs and reduce the range of symbols of each motif of length ℓ from range $[n]$ to the range $[1..\ell]$. This takes worst-case time $O(m \log\log r)$.

Recall that two or more elements of P of the same length and order-isomorphic should be associated with the same state in the $\mathcal{AC}$ automaton. In order to identify the order-isomorphic elements of P, we will carry a first step called normalization. It consists in normalizing each motif. A motif p is normalized by replacing each symbol p_j by the pair $\text{rep}(p = p_1 \dots p_{j-1}, j)$ (consisting in the positions of the predecessor and successor among symbols $p_1 \dots p_{j-1}$). This can be done for all motifs in total $O(m \log\log r)$ time. In the next step, we build a trie on the set of normalized motifs. This takes linear time. The trie naturally determines the forward transitions. More precisely any node in the trie will represent a state of the automaton and the labeled trie transitions will represent follow transitions.

Note that unlike the forward automaton (or the $\mathcal{MP}$ automaton) there could be more than one outgoing forward transition from each node. In order to encode the outgoing transitions from each node, we will make use of a hash table that stores all the transitions outgoing from that node. More precisely for each transition labeled by the pair $\text{rep}(p = p_1 \dots p_{j-1}, j)$ and directed to a state q, the hash table will associate the key p_1 associated with the value q. If we want to achieve complexity $O(\log d)$ per transition, then we organize the transition in a balanced binary search tree instead of a hash table. Now that the next transitions have been successfully built, the final step will be to build the failure transitions and this takes more effort. The construction of the failure transitions can also be done in worst-case $O(m \log\log r)$ time, but for lack of space we defer the details to the extended version [6].

We thus have the following theorem:

Theorem 3. *Building the $\mathcal{AC}$ automaton for a set of d consecutive motifs of total length m and where the longest motif is of length r can be done in worst-case $O(m \log\log r)$ time.*

5 Single Sublinear Average-Case Motif Searching

Algorithm forward takes $O(n + m \log\log m)$ time in the worst case but also on average. We present now a very simple and efficient average case-algorithm which takes $O(\frac{m \log m}{\log\log m} + n\frac{\log m}{m \log\log m})$ time.

In order to search for a motif p in t, we first build a tree T of all isomorphic-order factors of length $b = \lceil\frac{3.5 \log m}{\log\log m}\rceil$ of p^r (the reverse of p). T is built by inserting each such factor one after the other in a tree and building the corresponding path if it does not already exist. The construction of this tree requires $O(\frac{m \log m}{\log\log m})$ time (details are given below). The matching phase is performed through a window of size m that is shifted along the text. For each position of this window, b symbols are read backward from the end of the window in the tree T. Two cases may occur: *(i)* either the factor is not recognized as a factor of p^r. This means that no occurrence of p might overlap this factor and we can safely shift the search window past the first symbol of this factor;
(ii) or the factor is recognized, in which case we simply check if the motif is present using a naive $O(m)$ algorithm, and we repeat this test for the next $m-b$ symbols. This might require $O(m^2)$ steps in the worst case.
In both cases we then shift the window of $m - b + 1$ symbols.

Let us analyze the average complexity of our algorithm, in the following model: all text permutations are considered to be equiprobable, all integer values in a motif are distinct.

We count the average number of symbol comparisons required to shift the search window of $m-b+1$ symbols to the right. As there are $n/(m-b+1)$ such segments of length $m-b+1$ symbols in n, we will simply multiply the resulting complexity of the matching phase by $n/(m-b+1) = O(n/m)$ to get the whole average complexity of our algorithm (assuming T is already built).

There might be $O(b!)$ distinct motifs that could appear in the text while this number is bounded by $m-b+1$ in the motif (one by position). Thus, with a probability bounded by $\frac{m-b+1}{b!}$ we will recognize the segment of the text as a factor of p and enter case 2. In which case, moving the search window of $m-b+1=O(m)$ symbols to the right using the naive algorithm will require $O(m^2)$ worst case time.

In the other case which occurs with probability at least $1-\frac{m-b+1}{b!}$, shifting the search window by $m-b+1$ symbols to the right only requires reading b numbers.

The average complexity (in terms of number of symbol reading and comparisons) for shifting by $m-b+1$ symbols is thus (upper) bounded by $A = O((m^2)\frac{m-b+1}{b!} + b(1-\frac{m-b+1}{b!}))$ and the whole complexity by $O((n/m)A)$. By expanding and simplifying A we get that $A = O(b + O(m^3/b!))$. Now using the famous Stirling approximation $\ln(k!) = k\ln k - k + O(\ln k)$, it is not difficult to prove that $b! = 2^{b\log b - b\log e + O(\log b)} = \Omega(m^3)$ and thus $A = O(b)$ and the whole average time complexity (in terms of number of symbol reading and comparisons) turns out to be $O(\frac{n\log m}{m\log\log m})$.

Implementation Details. The tree T can actually be built in $O(\frac{m\log m}{\log\log m})$ time by using appropriate data structures. Recall that the tree T recognizes all the factors of p^r of length $\lceil\frac{3.5\log m}{\log\log m}\rceil$. To implement T, we use the same $\mathcal{AC}$ automaton presented in previous section to build the tree T, but with two differences: we only need forward transitions and the length of any motif is bounded by $\frac{\log m}{\log\log m}$. Thus the cost is upper bounded by $O(\frac{m\log m}{\log\log m}\cdot\tau)$, where τ is the time needed to do an operation on the predecessor data structure (maximum of the times needed for inserts/deletes and searches). We now turn our attention to the cost of the matching phase. From the previous section, we know that the total complexity in terms of number of symbol reading and comparisons is $O(\frac{n\log m}{m\log\log m})$. The total cost of the matching phase is dominated by the multiplication of the total number of text symbols read multiplied by the cost of a transition in the $\mathcal{AC}$ automaton which itself is dominated by $\tau = O(\min(\log\log n, \sqrt{\frac{\log r}{\log\log r}}, \log d))$, the time to do an operation on a predecessor data structure (or traversing a balanced binary search tree of size $O(d)$). The total cost of the matching phase is thus $O(\frac{n\log m}{m\log\log m}\cdot\tau)$.

Now the performance of both matching and building phases crucially depend on the used predecessor data structure. If a binary search tree is used then $\tau = O\left(\log\frac{\log m}{\log\log m}\right) = O(\log\log m)$ and the total matching time becomes $O(n\cdot\tau) = O(n\log\log m)$, and the total building time becomes $O(m\log m)$. However, we can do better if we work in the word-RAM model. Namely, we can use the atomic-heap (see Section 2) which would add additional $o(m)$ words of space and support all operations (queries, inserts and deletes) in constant time on structures of size $\log^{O(1)} m$. In our case, we have structures of maximal size $O(\frac{\log m}{\log\log m})$ and thus the operations can be supported in constant time. We thus have the following theorem:

Theorem 4. *Searching for a consecutive motif $p = p_1 \dots p_m$ in a permutation $t = t_1 \dots t_n$ can be done in average $O(\frac{m \log m}{\log \log m} + \frac{n \log m}{m \log \log m})$ time.*

6 Average Optimality

We prove in this section a lower bound on the average complexity of any consecutive motif matching algorithm. The proof of this bound is inspired by that of Yao [15] which proved an average lower bound for matching a (regular) motif of length m in a text of length n. We prove in our case of interest an average lower bound of $\Omega(\frac{n \log m}{m \log \log m})$ considering all permutations over $[n]$ to be equiprobable. As this average complexity is reached by the algorithm we designed in the previous section, this bound is tight.

We begin to circumscribe our problem on small segments of length $2m-1$ of the text into which we search for. Precisely, following [15,12], we divide our text in $\lfloor n/(2m-1) \rfloor$ contiguous and non-overlapping segments s_i, $1 \le i \le \lfloor n/(2m-1) \rfloor$, such that $s_i(t) = t_{(2m-1)(i-1)+1} \dots t_{(2m-1)i}$. When searching for a motif in t, there might be occurrences overlapping two blocks. But as we are interested in a lower bound, the following lemma allows us to focus on the inside of all segments.

Lemma 4. *A lower bound for finding a motif p inside all segments $s_i(t)$ is also a lower bound to the problem of searching for all occurrences of p in t.*

We now claim that instead of focusing on all segments $s_i(t)$, we can focus on obtaining a lower bound to search p in any single segment. Indeed these segments are non-overlapping and we are searching inside the segments.

Lemma 5. *The average time for searching for p inside all segments $s_i(t)$ is $\lfloor n/(2m-1) \rfloor$ times the average time for searching for p inside any such segment.*

Let $E(m)$ be the average complexity for searching a motif p of size m in any segment of size $2m-1$. Using the two previous lemmas, the whole average complexity is at least $\sum_{i=1}^{\lfloor n/(2m-1) \rfloor} E(m) = \lfloor n/(2m-1) \rfloor E(m) = \Omega(n/m)E(m)$.

It remains only to prove the lower bound $E(m) = \Omega(\frac{\log m}{\log \log m})$ to obtain the claimed lower bound for the whole problem.

Recall that we consider all $m!$ motif of size m to be equiprobable among the set S_m of permutations of length m. For $0 < \ell \le m$, let $\mathcal{P}_m(\ell)$ be the set of motifs of size m that can be searched using a sliding window of size m over a text of size $2m-1$ and checking only ℓ positions in this window. Then S_m is the disjoint union of $\mathcal{P}_m(\ell)$ and $S_m \setminus \mathcal{P}_m(\ell)$, that is the set of motifs that can be searched with only ℓ accesses and the others. For all motif in $\mathcal{P}_m(\ell)$, the average search complexity is counted 1 (lower bound). For any other motif in $S_m \setminus \mathcal{P}_m(\ell)$, the average search complexity is at least $\ell+1$. This leads to the following lemma:

Lemma 6. *For $0 < \ell \le m$, let $C(m, \ell) = \frac{|\mathcal{P}_m(\ell)| + (m! - |\mathcal{P}_m(\ell)|)(\ell+1)}{m!}$. Then $C(m, \ell)$ is a lower bound for the average complexity $E(m)$.*

We want now to maximize our bound in order to get a tight bound. To do so, we can choose ℓ depending on m.

Lemma 7. *There exists* $\ell(m)$ *s.t.* $0 < \ell(m) \leq m$ *and* $C(m, \ell(m)) = \Omega(\frac{\log m}{\log \log m})$.

We now sketch the proof of Lemma 7. As $C(m, \ell)$ decreases when $\mathcal{P}_m(\ell)$ increases, we search an upper bound for $\mathcal{P}_m(\ell)$. We prove that $|\mathcal{P}_m(\ell)| \leq m! \left(1 - \frac{1}{\ell!}\right)^{\left\lceil \frac{m-1}{\ell^2} \right\rceil}$ in the same way Yao proved the counting lemma of [15]. Thus we have $C(m, \ell) \geq \ell + 1 - \ell \cdot \left(1 - \frac{1}{\ell!}\right)^{\left\lceil \frac{m-1}{\ell^2} \right\rceil}$

We claim that $\ell = \frac{b \log m}{\log \log m}$ with $b = 1 + o(1)$ satisfies $98/100 \leq \left(1 - \frac{1}{\ell!}\right)^{\left\lceil \frac{m-1}{\ell^2} \right\rceil} \leq 99/100$ (Equation (E)). This gives $C(m, \ell(m)) \geq \ell + 1 - 98/100\ell = \Omega(\ell) = \Omega(\frac{\log m}{\log \log m})$, stating the lemma.

The idea to prove our claim :

Let us impose $\left\lceil \frac{m-1}{\ell^2} \right\rceil \times \frac{1}{\ell!} \leq 1/10$ (*ineq*.1). This allows us to approximate Equation (E) using the classical formula $(1 + x)^a = 1 + ax + \frac{a(a-1)}{2!}x^2 + \ldots + \frac{a!}{n!(a-n)!}x^n = 1 + ax + \gamma$ where $a = \left\lceil \frac{m-1}{\ell^2} \right\rceil$, $x = \frac{-1}{\ell!}$ and $\gamma = \sum_{i=2}^{n} \frac{a!}{i!(a-i)!}x^i$. It is easy to see that inequality (1) implies that γ converges and is dominated by its first term which is bounded $\frac{a(a-1)}{2!}x^2 \leq 1/200$. We thus deduce that $(1 + x)^a \in [1+ax, 1+ax+1/200]$ which implies that $(1+x)^a - 1/200 \leq 1+ax \leq (1+x)^a$. From $(1 + x)^a = \frac{|\mathcal{P}_m(\ell)|}{m!} \in [\frac{98}{100}, \frac{99}{100}]$, we obtain $\frac{98}{100} - \frac{1}{200} \leq 1 + ax \leq \frac{99}{100}$. By replacing a and x in $1+ax$ we get : $\frac{98}{100} - \frac{1}{200} = 195/200 \leq 1 - \left\lceil \frac{m-1}{\ell^2} \right\rceil \times \frac{1}{\ell!} \leq 99/100$. Then we prove that $\ell = \frac{b \log m}{\log \log m}$ with $b = 1 + o(1)$ satisfy these two last inequalities and inequality (1), implying that Equation (E) is satisfied.

Putting all the lemmas of this section together, we have that $\Omega(\frac{n \log m}{m \log \log m})$ is a lower bound of the whole average complexity for searching for a consecutive motif in a permutation.

References

1. AdelsonVelskii, M., Landis, E.M.: An algorithm for the organization of information. Defense Technical Information Center (1963)
2. Ahal, S., Rabinovich, Y.: On Complexity of the Subpattern Problem. SJDM 22(2), 629–649 (2008)
3. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Commun. ACM 18(6), 333–340 (1975)
4. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. J. ACM 54(3), 13 (2007)
5. Bayer, R.: Symmetric binary B-trees: Data structure and maintenance algorithms. Acta Informatica 1(4), 290–306 (1972)
6. Belazzougui, D., Pierrot, A., Raffinot, M., Vialette, S.: Single and multiple consecutive permutation motif search. CoRR, abs/1301.4952 (January 21, 2013)
7. Bose, P., Buss, J.F., Lubiw, A.: Pattern matching for permutations. Information Processing Letters 65(5), 277–283 (1998)
8. Kim, J., Eades, P., Fleischer, R., Hong, S.-H., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T.: Order preserving matching. arXiv preprint arXiv: 1302.4064 (2013)

9. Kitaev, S.: Patterns in Permutations and Words. In: EATCS. Springer (2011)
10. Kubica, M., Kulczyński, T., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for consecutive permutation pattern matching. Information Processing Letters (2013)
11. Morris Jr., J.H., Pratt, V.R.: A linear pattern-matching algorithm. Technical report, Univ. of California, Berkeley (1970)
12. Navarro, G., Fredriksson, K.: Average complexity of exact and approximate multiple string matching. TCS 321(2-3), 283–290 (2004)
13. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Inf. Process. Lett. 6(3), 80–82 (1977)
14. Willard, D.E.: Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. SIAM J. Comput. 29(3), 1030–1049 (1999)
15. Yao, A.C.: The complexity of pattern matching for a random string. SIAM Journal on Computing 8(3), 368–387 (1979)

Beating $\mathcal{O}(nm)$ in Approximate LZW-Compressed Pattern Matching*

Paweł Gawrychowski[1] and Damian Straszak[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
gawry@cs.uni.wroc.pl
[2] Institute of Computer Science, University of Wrocław, Poland
damian.straszak@gmail.com

Abstract. Given an LZW/LZ78 compressed text, we want to find an approximate occurrence of a given pattern of length m. The goal is to achieve time complexity depending on the size n of the compressed representation of the text instead of its length. We consider two specific definitions of approximate matching, namely the Hamming distance and the edit distance, and show how to achieve $\mathcal{O}(n\sqrt{m}k^2)$ and $\mathcal{O}(n\sqrt{m}k^3)$ running time, respectively, where k is the bound on the distance, both in linear space. Even for very small values of k, the best previously known solutions required $\Omega(nm)$ time. Our main contribution is applying a periodicity-based argument in a way that is computationally effective even if we operate on a compressed representation of a string, while the previous solutions were either based on a dynamic programming, or a black-box application of tools developed for uncompressed strings.

Keywords: approximate pattern matching, edit distance, Lempel-Ziv.

1 Introduction

Pattern matching, which is the question of locating an occurrence of a given pattern in a text, is the most natural task as far as processing text data is concerned. Virtually any programming language contains a more or less efficient procedure for solving this problem, and any text processing application, including the widely available `grep` utility, gives users the means of solving it. While exact pattern matching is well-understood, and in particular many linear time solutions are known [5], it seems that its approximate version is less understood. Two most natural versions of the question are pattern matching with errors, where one ask for a substring of the text with small edit distance to the pattern, and pattern matching with mismatches, where one is interested in a substring with small Hamming distance to the pattern. It is known that if N is the length of the text and k is the number of allowed errors or mismatches, both problems can be solved in $\mathcal{O}(Nk)$ time [10,11], and in fact the complexity for the latter version can be improved to $\mathcal{O}(N\sqrt{k \log k})$ [2]. Under the natural assumption that the value of k

* Supported by NCN grant 2011/01/D/ST6/07164, 2011–2014.

is small, one can do even better, and solve the problems in $\mathcal{O}(N + \frac{Nk^4}{m})$ [4] and $\mathcal{O}((N + \frac{Nk^3}{m}) \log k)$ [2] time complexity, respectively, which might be linear in N if k is small enough. Unfortunately, in some cases even a linear time complexity might be not good enough. This is the case when we are talking about large collections of repetitive data stored in a compressed form. Then the length of the text N might be substantially larger than the size n of its actual representation, and the goal is to achieve a running time depending on n, not N. Whether achieving such goal is possible clearly depends on the power of the compression method. In this paper we focus on the LZW/LZ78 compression [12,13], which is not as powerful as the more general LZ77 method, but still has some nice theoretical properties, and is used in real-world applications. It is known that exact LZW-compressed pattern matching can be solved very efficiently [1,6], even in the fully compressed version, where both the text and the pattern are LZW-compressed [8]. The obvious question is how efficiently can we solve approximate LZW-compressed pattern matching?

The best previously known solution by Kärkkäinen, Navarro, and Ukkonen [9], locates all occ occurrences with up to k errors using $\mathcal{O}(nmk+occ)$ time and $\mathcal{O}(nmk)$ space. More precisely, it outputs all ending positions j such that there is i for which the edit distance between $t[i..j]$ and p is at most k. In some cases, this time bound can be decreased using the idea of Bille, Fagerberg, and Gørtz [3], who presented a way to translate all uncompressed pattern matching bounds into the compressed setting. Their approach works for both the edit and Hamming distance, and by plugging the best known uncompressed pattern matching solutions, we can get:

1. $\mathcal{O}(nmk + occ)$ time and $\mathcal{O}(\frac{n}{mk} + m + occ)$ space for the edit distance,
2. $\mathcal{O}(nk^4 + nm + occ)$ time and $\mathcal{O}(\frac{n}{k^4+m} + m + occ)$ space for the edit distance,
3. $\mathcal{O}(n(k^3 + m) \log k + occ)$ time and $\mathcal{O}(\frac{n}{(k^3+m) \log k} + m + occ)$ space for the Hamming distance.

While the space complexity of the resulting algorithms is small, even for constant values of k the time complexity is $\Omega(nm)$, and in fact this is an inherent shortcoming of the approach: the best we can hope for is $\mathcal{O}(nm)$ for sufficiently small values of k, say, $k = \mathcal{O}(m^{1/3})$.

In this paper we show that in fact this barrier can be broken. We prove that for the Hamming distance, running time of $\mathcal{O}(n\sqrt{m}k^2)$ is possible, which for $k = o(m^{1/4})$ is $o(nm)$. Then we show how to extend the algorithm by building on the ideas of Cole and Hariharan [4], and achieve $\mathcal{O}(n\sqrt{m}k^3)$ for the edit distance. Both algorithms use $O(n + m)$ space. For the sake of clarity, we concentrate on the question of detecting just one occurrence, but our algorithms generalize to generating all of them.

Some of our methods are based on the concepts first used by Cole and Hariharan [4], and later by Amir, Lewenstein and Porat [2]. Applying them in the compressed setting is not just a trivial exercise, and creates new challenges. For instance, verifying whether a given position corresponds to an occurrence with no more than k mismatches in $\mathcal{O}(k)$ time is straightforward in the uncompressed setting using the suffix tree, but in our case requires some additional ideas.

We start with some basic tools in Sections 2. Then we distinguish between two types of matches, called internal and crossing. Detecting the former is relatively straightforward in both versions. To detect the latter, we reduce the question to a problem that is easier to work with, which we call pattern matching in pc-strings, see Section 3. To solve pattern matching with mismatches in pc-strings, we distinguish between two cases depending on how periodic the pattern is. For this we apply the concept of z-breaks, heavily used in the previous papers on approximate pattern matching. If there are many such breaks, or in other words the pattern is not very repetitive, we can solve the problem by reducing to a generalization of (exact) compressed pattern matching with multiple patterns, see Section 4. Otherwise, the pattern is *highly periodic*, and the situation is more complicated. In Section 5 we show how to exploit the regular structure of such pattern to construct an efficient algorithm. Then in Section 6, which is the most technical part of the paper, we speed up the method using a new technique which considers all candidates in a more global manner. Finally, in Section 7 we generalize the solution to solve the version with errors. Because of the space limitation, we omit many details, which can be found in the full version.

2 Preliminaries

We are given a text $t[1..N]$ and a pattern $p[1..m]$, both are strings over an integer alphabet Σ. We assume that $m \leq N$ and $\Sigma = \{1, 2, \ldots, N\}$. The pattern is given explicitly, but the text is described implicitly using the LZW/LZ78 compression scheme. Such scheme is defined as follows: we partition the text into n disjoint fragments $t = z_1 z_2 \ldots z_n$, where each fragment z_i is either a single letter, i.e., $z_i = c$, or a word of the form $z_i = z_j c$, where $j < i$. The fragments z_i are usually called the *codewords*, and because their set is closed under taking prefixes, we may represent it as a trie, which will be further denoted by T. Depending on how we choose the partition and encode the codewords, we get different concrete compression methods, say LZW or LZ78. Our methods do not depend on such technicalities as long as we are given T and the text is described as a list of pointers to the nodes of T representing the successive fragments.

The Hamming distance between two strings of the same length is simply the number of positions where their corresponding characters differ. The edit distance $\text{ed}(s, t)$ is the minimal number of operations necessary to transform s into t, where an operation is an insertion, replacement, or removal of a character.

The first problem we consider is *compressed pattern matching with mismatches*, where we are given a compressed representation of a text t, a pattern p, and a positive integer k. We want to find i such that the Hamming distance between $t[i..i + m - 1]$ and the pattern is at most k. We also consider *compressed pattern matching with errors*, where the goal is to find i and j such that the edit distance between $t[i..j]$ and p is at most k.

To efficiently operate on the compressed text and the pattern, we need a number of data structures. Given two subwords of the pattern s_1 and s_2 we can calculate their longest common prefix, denoted $\text{LCPref}(s_1, s_2)$, and longest common suffix, denoted $\text{LCSuf}(s_1, s_2)$, in constant time. Given i and j, we can retrieve $z_i[j]$ in constant time. Given a chunk s_1, where a chunk is a subword of some root-to-leaf path in T, and a subword of the pattern s_2, we can calculate $\text{LCSuf}(s_1, s_2)$ in constant time and $\text{LCPref}(s_1, s_2)$ in $\mathcal{O}(\log m)$. The total preprocessing time is $\mathcal{O}(n + m)$.

We need also some basic concepts from combinatorics on words. α is a period of a string s if $s[i] = s[i+\alpha]$ holds for every $i = 1, 2, \ldots, |s| - \alpha$, or in other words we can write $s = w^i u$, where $|w| = \alpha$ and $u \neq w$ is a prefix of w. The smallest such α is called **the** period of s. If the period of s is at most $\frac{|s|}{2}$, s is periodic, and otherwise we call it a break, or $|s|$-break. A word is primitive if it cannot be represented as a nontrivial power of some other word. For every word s, there exists its unique cyclic shift s' which is lexicographically smallest, and we call s' the cyclic representative of s. For a periodic s, the cyclic representative of w corresponding to the period of s is called the canonical period of s. One of the basic results concerning periods is the periodicity lemma, which says that if q and q' are both periods of s, and $q + q' \leq |s|$, so is $\gcd(q, q')$.

3 Further Preprocessing

From now on we fix k to be the number of allowed mismatches (errors) in our problem. We will say in short that the pattern matches at some position in the text if the Hamming distance (or the edit distance) between the pattern and the fragment of the text starting at this position is at most k. It is natural to distinguish between two types of matches: *internal matches* (the pattern lies fully within a single codeword) and *crossing matches* (the pattern crosses some boundary between two codewords). The internal matches can be efficiently generated using standard tools, and we focus on detecting the crossing matches, where the situation is much more complicated. In this case the pattern crosses at least one boundary between two codewords, and it may cross a lot of them, which seems hard to deal with. Anyway, it suffices to iterate over all $n - 1$ boundaries and for each of them find all matches that cross it. After fixing such a boundary, we may concentrate only on a window of length $2m$ containing m characters to the left and m to the right. Problems arise when there are many very short codewords in some fragment of the text, because in such a case all boundaries in this fragment will create windows containing lots of codewords. This is one of the obstacles we need to tackle to construct an efficient algorithm.

We want to make now one technical assumption, which simplifies significantly some definitions and the description of the algorithm. Namely, we will assume that each letter appearing in text, appears also in the pattern. Our algorithms work in the general case after minor modifications.

The notion of a pc-string will play the main role in the rest of the paper. Note that the definition changes slightly when we want to move from mismatches to errors. Nevertheless, the change is very small, so we prefer to have just one common definition, and keep in mind that its meaning depends on the variant.

Definition 1. *Let p be a pattern and f be a string. We say that $f = v_1 v_2 ... v_l$ is a pattern-compressed-string, in short pc-string, if:*

1. *$|f| \leq 2m$ ($|f| \leq 2m + 2k$ when we are dealing with errors) and $l \leq 4k + 5$,*
2. *v_i is a factor of p, for $i = 1, 2, \ldots, l$,*
3. *$v_i v_{i+1}$ is not a factor of p, for $i = 1, 2, \ldots, l - 1$.*

We represent such string as a list $(a_1, b_1), (a_2, b_2), ..., (a_l, b_l)$, where $v_i = p[a_i..b_i]$.

Pc-strings are very convenient to deal with. Because no $v_i v_{i+1}$ appears in p as a substring, we can answer any LCPref and LCSuf query between a subword of f and a subword of the pattern in constant time, as each result of such a query overlaps at most 3 v_i's, so we need at most 3 queries between factors of p.

Proposition 1. *Given a position in a pc-string f, we can verify whether the alignment of the pattern at this position results in a match in $\mathcal{O}(k)$ time.*

It turns out that finding matches crossing a fixed boundary can be reduced to one instance of pattern matching with mismatches or errors in a pc-string.

Theorem 1. *Suppose we have an algorithm solving pattern matching with k mismatches (errors) in pc-strings in $T_{PC}(m)$ time. Then we can solve pattern matching with k mismatches (errors) in LZW-compressed text in $\mathcal{O}(nk \log^2 m + m + n \cdot T_{PC}(m))$ ($\mathcal{O}(nk^2 + nk \log^2 m + m + n \cdot T_{PC}(m))$) time.*

4 Detecting Matches in Pc-Strings

In this section we concentrate on the version with mismatches and present an efficient algorithm for detecting matches in a pc-string. It will use a certain preprocessing of the pattern, which takes $\mathcal{O}(m)$ time and is performed just once in the whole solution, not every time we get a new pc-string.

We distinguish between two cases depending on the "level of periodicity" of the pattern. Let $z \geq 3$ be a parameter to be fixed later. We find in p as many disjoint z-breaks as possible, which can be done in $\mathcal{O}(m)$ time [4]. If there are just a few such breaks, the pattern can be seen as *highly periodic*. First we consider the opposite case when p contains at least $2k$ disjoint z-breaks. Then we can discard most of the starting positions, and verify all the remaining ones separately.

Lemma 1 (see [2]). *Let f be a text of length $2m$. Assume that the pattern p contains at least $2k$ disjoint z-breaks. Then there are at most $\mathcal{O}(\frac{m}{z})$ matches (with k mismatches) of p in f.*

Proof. Choose $2k$ disjoint occurrences of breaks in the pattern. Let $b_1, b_2, ..., b_r$ be all pairwise different breaks among them, with b_i occurring x_i times, so $\sum_{i=1}^{r} x_i = 2k$. Consider one break b_i, and denote the positions of the disjoint occurrences of b_i in p by $o_1, o_2, ..., o_{x_i}$. For each occurrence of b_i in the text, say at position q, we add a mark to all positions $q - o_1 + 1, q - o_2 + 1, \ldots, q - o_{x_i} + 1$ within the text. Since the distance between two different occurrences of b_i in the text is at least $\frac{z}{2}$

there will be at most $\sum_{i=1}^{r} x_i \frac{2m}{z} = \frac{4km}{z}$ marks. Consider now a position in the text where p matches with at most k mismatches. At least k of the $2k$ breaks have to match exactly, so we have at least k marks there. But there are only at most $\frac{4m}{z}$ positions with at least k marks. □

This lemma is very useful, but it does not give a method to find all these $\mathcal{O}(\frac{m}{z})$ positions. For this we need to locate all occurrences in f of up to $2k$ pattern breaks. We cannot simply use the usual multiple pattern matching algorithm, because it would cost $\Omega(m)$ time, which is too much. However, we know that there are at most $\mathcal{O}(\frac{km}{z})$ occurrences of these breaks in f. This fact, combined with an efficient algorithm for multiple pattern matching in a pc-string, which is an adaptation of the method of Gawrychowski [7], gives a solution.

Lemma 2. *We can preprocess the pattern and a collection of its disjoint z-breaks in $\mathcal{O}(m)$ time, so that later given any pc-string $f = v_1 v_2 ... v_l$ we can find all occ occurrences of the breaks in f in $\mathcal{O}(l \log m + occ)$ time.*

Theorem 2. *Suppose the pattern contains at least $2k$ disjoint z-breaks. Then pattern matching with k mismatches in pc-strings can be solved in $\mathcal{O}(k \log m + \frac{km}{z})$ time.*

Proof. First we find $2k$ disjoint z-breaks in the pattern. We want now to detect the at most $\mathcal{O}(\frac{m}{z})$ positions in f where p can potentially match. Proceeding as in the proof of Lemma 1, first choose some $2k$ disjoint z-breaks and find all their matches in f using the algorithm from Lemma 2. This costs us $\mathcal{O}(l \log m + occ) = \mathcal{O}(k \log m + \frac{km}{z})$ time. The marking phase can be done in $\mathcal{O}(\frac{km}{z})$ time. Now for each of the $\mathcal{O}(\frac{m}{z})$ positions verify whether p matches there in $\mathcal{O}(k)$ time. So we can find all matches of p in f in $\mathcal{O}(k \log m + \frac{km}{z})$ time. □

Choosing big z makes our algorithm really fast. However, the larger is z, the harder is for the pattern to contain many z-breaks. Furthermore, we cannot expect each pattern to have many z-breaks, even for small z. Therefore, we need a different algorithm for the case when p has few breaks, or is *highly periodic.* The algorithm has to take advantage of the regular structure of the pattern.

5 Basic Algorithm for Highly Periodic Patterns

In this section we assume the pattern is highly periodic. This means we can write it in the form $p = s_1 b_1 s_2 b_2 ... s_r b_r s_{r+1}$, where $r < 2k$, each b_i is a z-break and each s_i is a (possibly empty) string with period at most $\frac{z}{2}$. The fragments $s_1, s_2, ..., s_{r+1}$ are called *periodic stretches.* As in the previous section we are interested in finding a match (with at most k mismatches) of p in a pc-string f.

Below we describe how to reduce the general case to the one where the number of breaks in the text is small. A very similar reasoning can be also used in matching with errors, the only change being increasing some constants.

Lemma 3. *Suppose f is a string of length at most $2m$ and p is a pattern containing at most $2k$ disjoint z-breaks. There exists a subword f' of f having at most $6k+1$ disjoint z-breaks such that each match of p in f lies fully within f'. Moreover, such f' can be found in $\mathcal{O}(kz)$ time.*

By the discussion above we can restrict ourselves to pc-strings having at most $\mathcal{O}(k)$ disjoint z-breaks. We will give now an algorithm achieving $\mathcal{O}(zk^4)$ running time for pattern matching with k mismatches in such pc-strings. While this is not the best algorithm we have obtained, it serves well as an introduction to the more complicated $\mathcal{O}(zk^3)$ algorithm presented in the next section.

Let us summarize the situation. We are given a pattern of the form $p = s_1 b_1 ... s_r b_r s_{r+1}$ and a pc-string $f = s'_1 b'_1 ... s'_q b'_q s'_{q+1}$, where $r, q = \mathcal{O}(k)$, b's denote z-breaks and the periods of all s's are at most $\frac{z}{2}$. We will soon see that alignments of the pattern, where the pattern breaks and text breaks are not too close from each other, are nice to work with, so we handle the remaining ones separately.

Proposition 2. *There are at most $\mathcal{O}(zk^3)$ alignments of the pattern in the text such that some text break (or text endpoint) is within a distance of $z(k+1)$ from some pattern break (or pattern endpoint).*

In this (simple) version of the algorithm we just verify all these $\mathcal{O}(zk^3)$ positions in $\mathcal{O}(k)$ time per one. This results in $\mathcal{O}(zk^4)$ complexity and leaves us with the convenient case, where all distances between pattern and text breaks (or endpoints) are at least $z(k+1)$. We call such alignments *fine*, and we will soon see that a fine alignment resulting in a match has a very special structure.

Starting from now we assume that the distances between consecutive breaks in the text (and in the pattern) are at least $z(k+1)$, and otherwise group some breaks together. Our argument works also for such groups but we describe it just for breaks. Similarly, we want to assume that s_1 and s_{r+1}) are either empty or of length at least $z(k+1)$, so we extend the boundary breaks if needed.

One can easily see that there are at most $\mathcal{O}(k^2)$ intervals of consecutive fine alignments in the text. Within such an interval the order of appearance of the breaks does not change. Fix one interval and suppose we have at least one match there. We want to argue that in such a case all periodic stretches involved in this match are compatible, meaning that their canonical periods are identical, and moreover start with the same offset modulo the period.

Proposition 3. *Suppose w_1, w_2 are periodic strings with periods not exceeding $\frac{z}{2}$. If $w_1 \neq w_2$ and $|w_1| = |w_2| \geq z(k+1)$ then there are at least $k+1$ mismatches between these two words.*

Suppose there is a match at some fine alignment. Between two consecutive breaks (we consider here all pattern and text breaks) there is always a periodic portion of length at least $z(k+1)$. By Proposition 3, there must be a perfect match between the corresponding fragments. So in particular, the periods of the corresponding pattern periodic stretch and text periodic stretch agree. Considering the

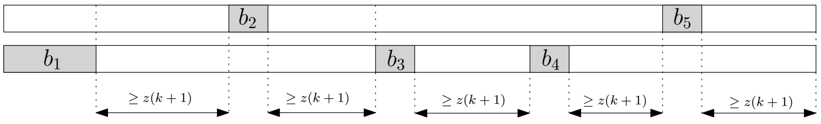


Fig. 1. Long overlaps between stretches imply their canonical periods are the same

way how the stretches overlap each other, see Figure 1, by transitivity all periodic stretches involved in the match have the same canonical period.

Suppose now all the periodic stretches in the pattern have the same canonical period u. We consider an interval of consecutive fine alignments. Assume there is a match somewhere in this interval. One can see that each two alignments i and $i + |u|$ from the interval have the same number of mismatches, because each break is aligned with a u-periodic stretch, so the fragment we compare it to is the same. So in order to find all matches within one interval, we only need to verify at most $|u| \leq \frac{z}{2}$ alignments. Each verification takes $\mathcal{O}(k)$ time, so the time taken over all intervals is $\mathcal{O}(k^2 \cdot \frac{z}{2} \cdot k) = \mathcal{O}(zk^3)$.

Theorem 3. *For highly periodic patterns, pattern matching with k mismatches in pc-strings can be solved in $\mathcal{O}(zk^4)$ time.*

6 Faster Algorithm for Highly Periodic Patterns

The purpose of this section is to show a faster algorithm for pattern matching with k mismatches in pc-strings, assuming the pattern is highly periodic. We will improve the time complexity from $\mathcal{O}(zk^4)$ to $\mathcal{O}(zk^3)$. We will make sure that the additional space required by the improved algorithm is just $\mathcal{O}(zk^2)$, which will be crucial in achieving linear space usage of the whole solution.

In the previous section we showed that one can assume that the text has at most $\mathcal{O}(k)$ disjoint z-breaks. The idea of the basic algorithm was to first work with the "bad" alignments. An alignment was considered "bad" if there was a text break and a pattern break close to each other (within a distance of $z(k+1)$). We took all such alignments and verified them in $\mathcal{O}(k)$ time each. The fine alignments (meaning not "bad") were analyzed in total time $\mathcal{O}(zk^3)$. This approach, although simple, seems to be very naive. Each time there is a single pair of close breaks, we waste $\Omega(k)$ time to deal with such an alignment. It turns out that we can verify a "bad" position in time proportional to the number of "bad" breaks. In the following definitions and lemmas we make the idea formal.

Definition 2. *In a fixed alignment of the pattern in the text, we call a pattern break black if there is some text break or text endpoint within distance $23zk$ from it. Similarly, we call a text break black if there is some pattern break or pattern endpoint within distance $23zk$ from it. Non-black breaks are called white.*

Note that one extreme case when a break is black is when it overlaps with some other break. It is convenient to deal with such situations separately. There are only

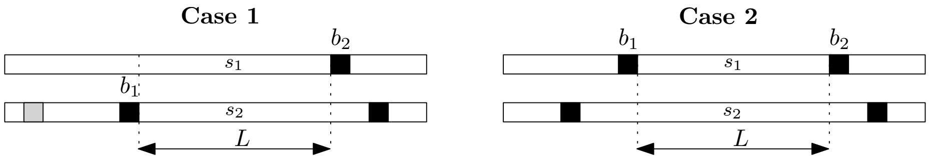


Fig. 2. Two consecutive black breaks

$\mathcal{O}(zk^2)$ such alignments, so they can be all verified in $\mathcal{O}(zk^3)$ time, and from now on we assume that no two breaks overlap. Moreover, we assume that there is at least one black break, as otherwise the alignment is fine.

Lemma 4. *After $\mathcal{O}(zk^3)$ time preprocessing, given an alignment with $B \geq 1$ black breaks we can test whether it corresponds to a match in $\mathcal{O}(B)$ time.*

We will prove the above lemma in the remaining part of this section. Suppose for a moment it holds, and consider all alignments with some black breaks. Call the number of black breaks in these alignments $B_1, B_2, ..., B_g$. Then by the above lemma, each single alignment can be processed in $\mathcal{O}(B_i)$ time, so the total time is $\mathcal{O}(\sum_{i=1}^{g} B_i)$. Every specific break is black at most $\mathcal{O}(k \cdot (46z(k+1)+2z)) = \mathcal{O}(zk^2)$ times, so $\mathcal{O}(\sum_{i=1}^{g} B_i) = \mathcal{O}(zk^3)$. So if we use this method to process the alignments, we will obtain an algorithm with $\mathcal{O}(zk^3)$ running time.

The main idea in the proof of the lemma is to partition the alignment into disjoint parts, such that in each of these parts we can count the number of mismatches easily. More precisely, if there are B black breaks in the considered alignment, we distinguish $\mathcal{O}(B)$ intervals where the Hamming distance can be determined in $\mathcal{O}(1)$ time, assuming some precalculation. We will now give the details by analyzing the relative arrangement of black and white breaks. Recall we have already reduced the situation to the case where no two breaks overlap.

Consider a periodic stretch s between two breaks in the pattern (text). It can be written in the form $s = u_1 u^i u_2$ where u is its canonical period (of length at most $\frac{z}{2}$), $i \geq 0$, u_1 is some suffix of u and u_2 is some prefix of u. Note also that the word u is primitive in such a case. It is easier to imagine the whole picture (and also to describe it) if $u_1 = u_2 = \varepsilon$, in other words when s is a power of its canonical period. We can achieve it by merging u_1 (u_2 respectively) to the neighboring break on the left (on the right). After this operation the breaks have lengths between z and $2z$ and all periodic stretches, maybe except these at the start and at the end of the word, are powers of primitive words.

Let us fix an alignment with at least one black break, and take any black pattern break (the reasoning for text breaks is the same). We want to count the number of mismatches between it and the corresponding periodic stretch from the text. To answer such a query in constant time, for each pattern break and periodic stretch u^i from the text we count mismatches between the break and the stretch for every possible shift smaller than $|u| \leq \frac{z}{2}$. Each such count can be performed in $\mathcal{O}(k)$ time, which results in $\mathcal{O}(zk^3)$ time preprocessing.

Now take two consecutive black breaks b_1, b_2. Consider the case, when there are no more breaks between them (of course there are no black ones, because we chose b_1, b_2 to be consecutive, but some white breaks might be there). Two possible situations are depicted in Figure 2. Our aim is now to count the number of mismatches between s_1 and s_2, which are length-L subwords of periodic stretches from the text and pattern, respectively. If $L \geq z(k+1)$ then by Proposition 3 either there are no mismatches between s_1 and s_2, or there are at least $k+1$ of them. It is easy to detect which case occurs: the strings agree if and only if their canonical periods are the same and they start with the same period offset, which can be determined in $\mathcal{O}(1)$ time after some straightforward preprocessing. So we can assume $L < z(k+1)$. We consider the cases from Figure 2 separately.

Case 1. In this case s_1 is length-L suffix of some text periodic stretch, s_2 is length-L prefix of some pattern periodic stretch. We want to precalculate all possible $\mathcal{O}(zk^3)$ results of such queries. Fix one pair of periodic stretches. We will calculate all the $\mathcal{O}(zk)$ required numbers in $\mathcal{O}(zk)$ total time. Let w be the canonical period of s_1, $d = |w|$ and let u be the canonical period of s_2. First calculate the answer for all overlaps of length at most d in $\mathcal{O}(dk) = \mathcal{O}(zk)$ time. Now to process an overlap of length $D > d$, we use the result for $D - d$, and add the number of mismatches between w and some factor of an infinite word u^∞, which can be previously precomputed in $\mathcal{O}(|u|k) = \mathcal{O}(zk)$ total time. Hence we can precalculate all values in $\mathcal{O}(zk^3)$ time, but space usage of $\mathcal{O}(zk^3)$ is too high to achieve linear total space complexity. It can be reduced to $\mathcal{O}(z)$ per a pair of stretches by carefully arranging some partial results so that the final answer can be computed as a difference of their prefix sums.

Case 2. In this case s_1 is a complete periodic stretch, and s_2 is a factor of a periodic stretch. Note that if s_2 has period d then there are only d essentially different alignments of such form. Overall there are only $\mathcal{O}(zk^2)$ possible queries, so we precalculate all of them in $\mathcal{O}(zk^3)$ time.

Then we need to consider the general situation when there are some white breaks between two consecutive black breaks b_1, b_2. Using a similar (although more complex) reasoning it can be solved in constant time after $\mathcal{O}(zk^2)$ space and $\mathcal{O}(zk^3)$ time preprocessing. Hence whenever we have an alignment with B black breaks, we may partition it into $\mathcal{O}(B)$ regions and either count the mismatches in each of them, or report that it exceeds k, in constant time, thus the theorem.

Theorem 4. *For highly periodic patterns, pattern matching with k mismatches in pc-strings can be solved in $\mathcal{O}(zk^3)$ time using $\mathcal{O}(zk^2)$ additional space.*

It is now a good moment to specify z. Let $z = \frac{\sqrt{m}}{k}$. Using Theorem 2 and Theorem 4 we see that such a choice of z gives us a running time $\mathcal{O}(k \log m + \sqrt{m}k^2) = \mathcal{O}(\sqrt{m}k^2)$ for pattern matching with k mismatches in pc-strings. The additional space needed is $\mathcal{O}(zk^2) = \mathcal{O}(\sqrt{m}k)$. By Theorem 1 we then obtain that pattern matching with k mismatches in LZW-compressed text can be solved in $\mathcal{O}(nk \log^2 m + m + n\sqrt{m}k^2)$, which is $\mathcal{O}(n\sqrt{m}k^2)$ because $n \geq \sqrt{m}$. The space complexity is

$\mathcal{O}(n+m+\sqrt{m}k)$. This is bounded by $\mathcal{O}(n+m)$ whenever $k = \mathcal{O}(\sqrt{m})$. In the opposite case we use the $\mathcal{O}(mk)$ algorithm [10] to process each pc-string using $\mathcal{O}(n+m)$ space and $\mathcal{O}(nmk) = \mathcal{O}(n\sqrt{m}k^2)$ total time.

Theorem 5. *Pattern matching with k mismatches in LZW-compressed strings can be solved in $\mathcal{O}(n\sqrt{m}k^2)$ time and $\mathcal{O}(n + m)$ space.*

7 Algorithm for Pattern Matching with Errors

In this section we discuss the algorithm for pattern matching with k errors in pc-strings. It is obtained by combining our methods for compressed strings (applied for pattern matching with mismatches) with the ideas used by Cole and Hariharan [4]. We need $\mathcal{O}(\frac{mk^2}{z} + k \log m)$ time for the case when p has at least $2k$ disjoint z-breaks and $\mathcal{O}(zk^4)$ for the case when p has less than $2k$ disjoint z-breaks. Choosing z to be $\frac{\sqrt{m}}{k}$ we obtain the following result.

Theorem 6. *Pattern matching with k errors in LZW-compressed strings can be solved in $\mathcal{O}(n\sqrt{m}k^3)$ time and $\mathcal{O}(n + m)$ space.*

References

1. Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. J. Comput. Syst. Sci. 52(2), 299–307 (1996)
2. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with k mismatches. J. Algorithms 50(2), 257–275 (2004)
3. Bille, P., Fagerberg, R., Gørtz, I.L.: Improved approximate string matching and regular expression matching on Ziv-Lempel compressed texts. ACM Transactions on Algorithms 6(1) (2009)
4. Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. SIAM J. Comput. 31(6), 1761–1782 (2002)
5. Crochemore, M., Rytter, W.: Jewels of stringology. World Scientific (2002)
6. Gawrychowski, P.: Optimal pattern matching in LZW compressed strings. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 362–372. SIAM (2011)
7. Gawrychowski, P.: Simple and efficient LZW-compressed multiple pattern matching. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 232–242. Springer, Heidelberg (2012)
8. Gawrychowski, P.: Tying up the loose ends in fully LZW-compressed pattern matching. In: Dürr, C., Wilke, T. (eds.) STACS. LIPIcs, vol. 14, pp. 624–635. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
9. Kärkkäinen, J., Navarro, G., Ukkonen, E.: Approximate string matching on Ziv-Lempel compressed text. J. Discrete Algorithms 1(3-4), 313–338 (2003)
10. Landau, G.M., Vishkin, U.: Efficient string matching with k mismatches. Theor. Comput. Sci. 43, 239–249 (1986)
11. Landau, G.M., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms 10(2), 157–169 (1989)
12. Welch, T.A.: A technique for high-performance data compression. Computer 17(6), 8–19 (1984)
13. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 24(5), 530–536 (1978)

Less Space: Indexing for Queries with Wildcards

Moshe Lewenstein[1,⋆], J. Ian Munro[2,⋆⋆], Venkatesh Raman[3,⋆⋆⋆], and Sharma V. Thankachan[4,†]

[1] Bar-Ilan University, Israel
moshe@macs.biu.ac.il
[2] University of Waterloo, Canada
imunro@uwaterloo.ca
[3] The Institute of Mathematical Sciences, India
vraman@imsc.res.in
[4] Louisiana State University, USA
thanks@csc.lsu.edu

Abstract. Text indexing is a fundamental problem in computer science, where the task is to index a given text (string) $T[1..n]$, such that whenever a pattern $P[1..p]$ comes as a query, we can efficiently report all those locations where P occurs as a substring of T. In this paper, we consider the case when P contains wildcard characters (which can match with any other character). The first non-trivial solution for the problem is given by Cole et al. [STOC 2004], where the index space is $O(n \log^k n)$ words or $O(n \log^{k+1} n)$ bits and the query time is $O(p + 2^h \log\log n + occ)$, where k is the maximum number of wildcard characters allowed in P, $h \leq k$ is the number of wildcard characters in P and occ represents the number of occurrences of P in T. Even though many indexes offering different space-time trade-offs were later proposed, a clear improvement on this result is still not known. In this paper, we first propose an $O(n \log^{k+\epsilon} n)$ bits index achieving the same query time as that of Cole et al.'s index, where $0 < \epsilon < 1$ is an arbitrary small constant. Then we propose another index of size $O(n \log^k n \log \sigma)$ bits, but with a slightly higher query time of $O(p + 2^h \log n + occ)$, where σ denotes the alphabet set size.

1 Introduction and Related Work

Text indexing is a fundamental problem in computer science, where the task is to index a given text (string) $T[1..n]$, such that whenever a pattern $P[1..p]$ comes as

⋆ The author is grateful for the support of the Binational Science Foundation (BSF) grant # 2010437 and for the support of the German Israel Foundation (GIF) grant # 1147/2011. This work was done while the author was on sabbatical at U. Waterloo, Canada.

⋆⋆ This research was funded in part by NSERC of Canada and the Canada Research Chairs program.

⋆⋆⋆ This work was done while the author was on sabbatical at U. Waterloo, Canada.

† Supported by National Science Foundation (NSF) Grants CCF–1017623 (R. Shah and J. S. Vitter) and CCF–1218904 (R. Shah). Part of this work was done while the author was visiting U. Waterloo, Canada.

a query, we can efficiently report all those locations where P occurs as a substring of T. The classic data structures for solving this problem are suffix trees [28] and suffix arrays [21]. Both these linear space ($O(n \log n)$ bits) structures can perform pattern matching in optimal $O(p + occ)$ and $O(p + \log n + occ)$ time respectively, where occ is the number of occurrences of P in T [1]. Approximate string matching and wildcard matching are natural extensions of the pattern matching problem. Both have been studied extensively. [2,11,15,8,18,26,27,14,6,19,20]. These problems have several applications in information retrieval, bioinformatics, data mining, and internet traffic analysis [7,13].

The focus of this paper is on the following problem: index T for handling matching of a query pattern P with at most k wildcards. A wildcard, also known as don't care character (represented by ϕ) can match with any other character in the alphabet set Σ (of size σ). Therefore, the pattern P can be written as $P_0\phi P_1\phi..P_{h-1}\phi P_h$, the concatenation of substrings $P_0, P_1, ...P_{h-1}, P_h$ separated by ϕ and $h \leq k$ is the number of wildcards in P. The first non-trivial solution for this problem was proposed by Cole et al. [11], where the index space is $O(n \log^k n)$ words or $O(n \log^{k+1} n)$ bits and query time is $O(p + 2^h \log \log n + occ)$. Recently, Bille et al. [6] proposed an index, which is a generalization of Cole et al.'s index. The space and query time are $O(n \log n \log_\beta^{k-1} n)$ words and $O(p+\beta^h \log \log n+occ)$ respectively, where $2 \leq \beta \leq \sigma$. Note that Cole et al.'s [11] result can be obtained by substituting $\beta = 2$. Bille et al. [6] also proposed an optimal $O(p + occ)$ time index of space $O(n\sigma^{k^2} \log^k \log n)$ words. Another space-efficient index of $O(n \log n)$ words proposed by Cole et al. [11] can answer this query in $O(p + \sigma^h \log \log n + occ)$ time, and is recently improved to $O(n)$ words without affecting the query time [6]. Several other linear space structures also exist in literature, such as the ones by Iliopouls and Rahman [22], and Lam et al. [18]. However, these indexes take $\Theta(nh)$ worst case time for answering the query. Despite all these continued efforts, a clear improvement over the seminal result by Cole et al. [11] (i.e., $O(n \log^{k+1} n)$ bits and $O(p + 2^h \log \log n + occ)$ time) is still not known.

In this paper, we describe two results. The first one is an $O(n \log^{k+\epsilon} n)$ bits index with $O(p + 2^h \log \log n + occ)$ query time, where where $0 < \epsilon < 1$ is an arbitrary small constant. The second one is an $O(n \log^k n \log \sigma)$ bits index, but with a slightly worse query time of $O(p + 2^h \log n + occ)$, where $\Sigma = [\sigma]$ denotes the alphabet set. Notice that our first result is a clear improvement over the earlier result by Cole et al., whereas the second one provides another space-time trade-off for this problem when the alphabet set is small.

Another problem that is strongly connected to the problem under consideration is to index the text wildcards. This was solved in Cole et al. [11] as well. However, for this case, better solutions have appeared in a succession of papers and indexes with succinct space and competitive query time [18,26,27,14] are available in the literature. Yet another related problem is that of indexing with gaps. Gaps are essentially longer wildcards. In [16] an index was proposed supporting queries of patterns containing one gap with a predefined length. This

[1] All logarithms in this article are base 2.

result builds on the result of [2]. The case of one gap was further improved by Bille et al. [5] with optimal query time. In [19] results were shown for the case when there is a larger number of gaps.

Outline. Section 2 gives the preliminaries. Next, we describe a classical framework for the case where $k = 1$ and then the framework by Cole et al.'s for $k \geq 1$ in Section 3, and Section 4, respectively. Section 5 describes our space-efficient data structures.

2 Preliminaries

2.1 Suffix Trees and Suffix Arrays

Suffix trees [28] and suffix arrays [21] are two classic data structures for online pattern matching queries. For a text $T[1..n]$, substring $T[i..n]$, with $i \in [1, n]$, is called a suffix of T. The suffix tree for T is a lexicographic arrangement of all these n suffixes in a compact trie structure, where the i^{th} leftmost leaf represents the i^{th} lexicographically smallest suffix. For each node v in the suffix tree, we use $path(v)$ to denote the concatenation of edge labels along the path from the root to v. For any pattern P (of length p), the locus of P in the suffix tree is defined to be the highest node v (i.e., the closest node from the root) such that P is a prefix of *path(v)* and can be computed in $O(p)$ time.

The suffix array $SA[1..n]$ is an array of length n, such that $SA[i]$ is the starting position of the i^{th} lexicographically smallest suffix of T. The suffix array has an important property that the starting positions of all suffixes with the same prefix are always stored in a contiguous region in SA. Based on this property, the suffix range of a pattern P in SA is defined as the the maximal range $[sp, ep]$ such that for all $j \in [ep, ep], SA[j]$ is the starting point of a suffix of T with P as a prefix. In other words, the suffix range of a string represents the set of leaves in the subtree of its locus node in suffix tree. We also define its inverse, SA^{-1} to be an array such that $SA[i] = j$ if and only if $SA^{-1}[j] = i$. Both suffix trees and suffix arrays (along with an auxiliary data structure called LCP array) take $(n \log n)$ bits space and can perform pattern matching in optimal $O(p + occ)$ and $O(p + \log n + occ)$ time respectively, where occ is the number of occurrences of P in T.

2.2 Heavy Path and Heavy Path Decomposition

Let $\mathcal{T}$ be a tree with n nodes. We define the *size* of an internal node v to be the number of leaves in the subtree rooted at v. Then the *heavy path* of the tree $\mathcal{T}$ is the path starting from the root, where each node v on the path is the largest-size child of its parent. The *heavy path decomposition* of the tree $\mathcal{T}$ is the operation where we decompose each off-path subtree of the heavy path recursively; as a result, the edges in $\mathcal{T}$ will be partitioned into disjoint heavy paths. In [25], Sleator and Tarjan proved that the path from the root of $\mathcal{T}$ to any node v traverses at most $\log n$ heavy paths.

2.3 Two-Dimensional Orthogonal Range Reporting

Let $\mathcal{R} = \{(x_1, y_1), (x_2, y_2), .., (x_n, y_n)\}$ be a set of n points in an $[1, n] \times [1, n]$ grid. Without loss of generality, we assume that $x_i \leq x_{i+1}$. An orthogonal range reporting query on $\mathcal{R}$ is defined as follows: Given a query range $[x', x''] \times [y', y'']$, report all points (x_i, y_i) such that $x_i \in [x', x'']$ and $y_i \in [y', y'']$. Such a query can be answered optimally in $O(\log\log n + occ)$ time using an $O(n \log^{\epsilon} n)$-word space structure, where $\epsilon > 0$ is any arbitrary small constant [1]. See [20,10] for connections between text indexing and range searching.

2.4 Partial Rank Queries

Let $E[1..n]$ be an array of n characters taken from an alphabet set $\Sigma = [\sigma]$. Then $rank_E(i, c)$ where $c \in \Sigma$ is defined as the number of occurrences of c in $E[1..i]$. There exists $n \log \sigma + o(n \log \sigma)$-bit representations of E which can answer *rank* queries in $O(\log\log \sigma)$ time [12]. Rank queries of the type $rank_E(i, E[i])$ (or simply $prank_E(i)$) are called partial rank queries (also known as special rank queries [17]), and can be supported in constant time by maintaining an additional $o(n \log \sigma)$ bits structure [3,4].

3 The Classical Framework for $k = 1$

In this section, we describe a simple index for pattern matching with exactly one wildcard character. In this case, P can be written as $P_0 \phi P_1$, where P_0 and P_1 are the longest prefix and suffix respectively of P which do not contain any wildcard. The index is based on the following idea by Amir et al. [2]: if there exists an occurrence of P in T with the wildcard character ϕ matching exactly at the location $i \in [1, n]$ in T, then P_0 must be a suffix of $T[1..i-1]$ and P_1 must be a prefix of $T[i+1..n]$. All such i's can be quickly computed by maintaining the following structures:

1. Suffix tree of T (ST)
2. Suffix tree of T^R (RST), where T^R is the reverse of T. i.e., $T^R[i] = T[n-i+1]$.
3. A two-dimensional orthogonal range reporting structure (RR2D) over a set of n points of the form (x_i, y_i), where x_i is the lexicographic rank of $T[i+1..n]$ among all suffixes of T, and y_i is the lexicographic rank of $T[1..i-1]^R$ among all suffixes of T^R.

The index space can be bounded by $O(n \log^{\epsilon} n)$ words, where ST and RST takes $O(n)$-word space and RR2D structure (Section 2.3) takes $O(n \log^{\epsilon} n)$-word space. The query corresponding to an input $P = P_0 \phi P_1$ can be answered as follows: first find the suffix range $[sp, ep]$ of P_1 in ST, and the suffix range $[sp', ep']$ of P_0^R in RST in $O(|P_0| + |P_1|)$ time. Then, we issue a 2-dimensional orthogonal range reporting query on *RR2D* structure with $[sp, ep] \times [sp', ep']$ as the query range. The required time will be $O(\log\log n)$ plus the number of outputs. Corresponding to each point (x_j, y_j) reported as an output, there exists a match of P in T at the position $j - |P_0|$. Putting everything together, the total query time can be bounded as $O(|P_0| + |P_1| + \log\log n + occ)$. Bille et al. [5] showed that the $\log\log n$ additive factor in time can be removed by maintaining an $O(n \log\log n)$-word and optimal query time structure for $p < \log\log n$.

Theorem 1. *A given text $T[1..n]$ can be indexed in $O(n \log^\epsilon n)$ words, and all occurrences of a query pattern $P[1..p] = P_0 \phi P_1$ can be retrieved in $O(p + occ)$ time, where $0 < \epsilon < 1$ is an arbitrary small constant.*

Unfortunately, this approach cannot be generalized for $k \geq 2$.

4 Cole et al.'s Framework

In this section, we briefly describe the structure (we name it as STR_k) by Cole et al. [11] for handling pattern matching with at most k number of wildcards. The exact pattern matching problem (i.e., $k = 0$) can be answered using a suffix tree data structure, and for consistency we denote the suffix tree of T by STR_0. We shall call the nodes in STR_0 as *level*-0 *nodes.* The structure STR_k can be constructed in a recursive manner. We start with the description of STR_1, which is essentially an STR_0 with each of its nodes augmented with a compact trie called a side tree as follows: for every node u in STR_0 (i.e., level-0 nodes), with v being a child on the same heavy path as that of u, we choose all suffixes in the subtree of u [2], but not in the subtree of v, delete their first $|Path(u)| + 1$ characters [3] and maintain them as a compact trie. We call this compact trie as the side tree of u and is represented by $Sidetree(u)$. Then u is connected to the root of $Sidetree(u)$ via an edge with label ϕ (we fix the root of $Sidetree(u)$ as the last child of u). We now call a node a *level*-1 *node*, if it belongs to any *Sidetree* associated with a level-0 node. Using the same procedure as described above for constructing side trees from level-0 nodes, we construct side trees from level-1 nodes and call the newly formed nodes as *level*-2 *nodes.* Then we construct side trees from level-2 nodes and obtain *level*-3 *nodes* as so on until *level*-k *nodes.* The number of level-j nodes is given by $O(n \log^j n)$, therefore STR_k consists of $O(n \sum_{j=1}^{k} \log^j n) = O(n \log^k n)$ nodes and it can be maintained in $O(n \log^k n)$ words or $O(n \log^{k+1} n)$ bits. For every node u in STR_k, $path(u)$ represents the concatenation of edge labels on the path from the root of STR_k to u. Let ℓ_i represents the ith leftmost leaf node in STR_k. Notice that $path(\ell_i)$ corresponds to a suffix of T and we use $pos(\ell_i)$ to denote the starting position of that suffix[4]. Moreover if ℓ_i is a level-0 node, then $path(\ell_i) = T[pos(\ell_i)..n]$, whereas if it is a level-j node for $j \in [1, k]$, then $path(\ell_i)$ is given by $T[pos(\ell_i)..n]$ with its j characters replaced by ϕ.

Now a query corresponding to a pattern $P = P_0 \phi P_1 \phi .. \phi P_h$ can be answered as follows: start navigating the structure STR_k from its root by matching the

[2] This means all suffixes corresponding to the leaves in the subtree of u.

[3] which is the same as removing $|Path(u)| + 1$ characters from the prefix of all those suffixes, yet again, a collection of suffixes

[4] In the case of suffix tree STR_0, $pos(\ell_i) = SA[i]$.

characters in P one by one. Note, because ϕ is a wildcard it can match with any other character. However, if we have reached up to a node u in STR_k by matching a prefix of P and the next character to be matched is ϕ, by continuing to match ϕ with any other character will branch out the search into $degree(u)$ paths, where $degree(u) \leq \sigma + 1$ represents the number of outgoing edges from u. Cole et al. [11] observed that instead of matching in $degree(u)$ paths, it is enough to take only the following two paths (i) the outgoing path from u with its first character being ϕ and (ii) the heavy path on which u is sitting. Thus due to a single wildcard, the query will branch out to two paths, and in general for h wildcards, query will branch out to at most 2^h paths, ending up in $O(2^h)$ locus nodes. However, the time required for finding those $O(2^h)$ locus nodes is $O(|P_0| + 2|P_1| + 4|P_2| + ... + 2^h|P_h|) = O(2^h p)$. Using some auxiliary data structures, which are called *LCP data structures* occupying $O(n \log^{k+1} n)$ bits, this time complexity can be improved to $O(p + 2^h \log\log n)$. Then for every leaf ℓ_i in the subtree of a locus node, $pos(\ell_i)$ represents an occurrence of P in T. Thus all occurrences can be reported by spending another $O(occ)$ time.

Theorem 2. *([11]) A given text T of length n can be indexed in $O(n \log^{k+1} n)$ bits, such that all those occ occurrences of a pattern P containing $h \leq k$ wildcards can be reported in $O(p + 2^h \log\log n + occ)$ time.*

4.1 Finding Locus Nodes without *LCP Data Structures*

Even without the *LCP data structures*, the locus nodes can be computed efficiently using an $O(p + 2^h \log n)$ time algorithm. We start with the following definition: let $loc(u, d)$ refers to the location on the path from the root of STR_k to node u, such that the string obtained by concatenating edge labels on the path from the root of STR_k to $loc(u, d)$ (denoted by $path(loc(u, d))$) is the prefix of $path(u)$ of length $|path(u)| - d$. Notice that, $loc(u, 0)$ refers to node u itself. The maximum value of d for a particular node u is restricted by the following condition that there exits no other node on the path from $loc(u, d)$ to u. We now prove the following result.

Lemma 1. *Let $loc(u', d')$ represents a location in STR_k, which can be reached if we start matching a pattern P' from the location $loc(u, d)$. Then, given $loc(u, d)$ and the suffix range $[L', R']$ of a pattern P' in the suffix tree STR_0, we can find $loc(u', d')$ (if it exists) in $O(\log n)$ time.*

Proof. Let $\{\ell_i | i \in [x, y]\}$ and $\{\ell_i | i \in [x', y']\}$ represent the set of leaves in the subtree of u and u' respectively. Notice that $x \leq x' \leq y' \leq y$. Since the first $|path(loc(u, d))|$ characters are the same for all strings corresponding to $path(\ell_i)$ for $i \in [x, y]$, the lexicographic ordering among these strings will remain unchanged even if we remove their first $|path(loc(u, d))|$ characters. This means the function $SA^{-1}[pos(\ell_i) + |path(loc(u, d))|]$ is monotonically increasing with respect to $i \in [x, y]$. Moreover their next $|P'|$ characters match with P' iff $SA^{-1}[pos(\ell_i) + |path(loc(u, d))|] \in [L', R']$. Therefore x' and y' are the minimum and the maximum values of j satisfying this condition respectively, and they can be computed in $O(\log n)$ time using a binary search. Once x' and y' have been identified, u' can be computed in $O(1)$ by taking the lowest common ancestor of $\ell_{x'}$ and $\ell_{y'}$, and d' is given by $|path(u')| - |path(u)| + d' - |P'|$. Notice that $|path(\cdot)|$ for every node can be stored explicitly without changing the space bounds. □

Using the above result, we can compute the locus nodes as follows: for $i = 0, 1, 2, ..., h$, find the suffix ranges $[sp_i, ep_i]$ of P_i in the suffix tree STR_0 in overall $O(p)$ time. Now start navigating STR_k from its root by matching the characters of P_0. Whenever the query branch out to two paths, and if the next character to be matched is a wildcard character, it takes $O(1)$ per match. After that if we want to match the next P_i characters for some $i \in [1, h]$, we simply use the result in Lemma 1. Therefore, total time for pattern search can be bounded by $O(p + \log n + 2\log n + 4\log n + ... + 2^h \log n) = O(p + 2^h \log n)$. By putting every thing together, we have the following result.

Lemma 2. *There exists an $O(p + 2^h \log n)$ time algorithm for finding the locus nodes of P in STR_k.* □

5 New Space-Efficient Indexes

5.1 An $O(n \log^{k+\epsilon} n)$-bit Index

This result is achieved by a simple combination of the classical framework and Cole et al.'s framework. If there exists an occurrence of P in T with the first wildcard character ϕ matching exactly at the location $i \in [1, n]$ in T, then P_0 must be a suffix of $T[1..i-1]$ and $P_1\phi..\phi P_h$ must be a prefix of $T[i+1..n]$. All such i's can be quickly computed by maintaining the following structures:

- Cole et al.'s structure (STR_{k-1} of space $O(n \log^k n)$ bits) for handling the case only up to $k-1$ wildcards (along with the LCP data structures). The number of nodes in this structure is $O(n \log^{k-1} n)$. Here we use ℓ_i to denote the ith leftmost leaf in STR_{k-1}.
- Suffix tree of T^R (RST).
- Let L_i represents the set of leaves in STR_{k-1} with its $pos(\cdot) = i+1$ and let i' be the lexicographic rank of $T[1..i-1]^R$ among all suffixes of T^R. Construct the set S_i of two dimensional points (j, i') corresponding to each leaf $\ell_j \in L_i$. Note that $|L_i| = |S_i| = O(\log^{k-1} n)$. We then maintain an orthogonal range reporting structure RR2D (refer to section 2.3) over a set $\cup_{i=2}^{n-1} S_i$ of $O(n \log^{k-1} n)$ two dimensional points. The space required for this component is $O(n \log^{k+\epsilon} n)$ bits.

Now the pattern matching query can be answered as follows: if $h \leq k-1$, the query can be answered using STR_{k-1} in $O(p + 2^h \log\log n + occ)$ time. If $h = k$, we spilt the pattern P into $P_{suf} = P_1\phi..\phi P_h$ and $P_{pre} = P_0^R$. Then, search for P_{suf} in STR_{k-1} and compute $O(2^{h-1})$ locus nodes $u_P^1, u_P^2, u_P^3, ..$ and their corresponding suffix ranges $[L_1, R_1], [L_2, R_2], [L_3, R_3], ..$ etc (here ℓ_{L_z} and ℓ_{R_z} represents the leftmost and the rightmost leaves in the subtree of u_P^z) in $O(p+2^h \log\log n)$ time (using *LCP* data structures). Then search for P_{pre} in RST and obtain the suffix range $[sp', ep']$. Finally the occurrences can be computed by issuing $O(2^{h-1})$ two-dimensional range reporting queries on RR2D corresponding to the ranges $[L_1, R_1] \times [sp', ep'], [L_2, R_2] \times [sp', ep'], [L_3, R_3] \times [sp', ep']$... It can be easily verified that for every point $(j, .)$ reported as an output by the structure, there exists an occurrence of P_{suf} starting at the location $pos(\ell_j)$ and an occurrence of P_{pre} ending at the location $pos(\ell_j) - 2$ in T. Hence an occurrence of P at the location $pos(\ell_j) - |P_0| - 1$. By combining the above pieces, we have the following theorem.

Theorem 3. *A given text T of length n can be indexed in $O(n \log^{k+\epsilon} n)$ bits, such that all those occurrences of a pattern P containing $h \leq k$ wildcards can be retrieved in $O(p + 2^h \log\log n + occ)$ time, where $0 < \epsilon < 1$ is an arbitrary small constant.* □

By using an alternative RR2D structure of $O(n)$-word space with query time $O((1 + output) \log^{\epsilon} n)$ [9], we can obtain another space-time trade-off as follows:

Corollary 1 *A given text T of length n can be indexed in $O(n \log^k n)$ bits, such that all those occurrences of a pattern P containing $h \leq k$ wildcards can be retrieved in $O(p + (2^h + occ) \log^{\epsilon} n)$ time, where $0 < \epsilon < 1$ is an arbitrary small constant.*

Remark. Our techniques can be combined with the result by Bille et al. [6], and an $O(n \log^{1+\epsilon} n \log_{\beta}^{k-2} n)$-word index with $O(p + \beta^{h-1} \log\log n + occ)$ query time can be obtained, where $0 < \epsilon < 1$ is an arbitrary small constant and $2 \leq \beta \leq \sigma$.

5.2 An $O(n \log^k n \log \sigma)$-bit Index via Side Tree Compression

First we maintain the structure STR_{k-1} (as described before) in $O(n \log^k n)$ bits space. Therefore, the string matching case where the number of wildcards is at most $k-1$ can be handled efficiently. In order to handle the k-wildcard case (i.e., $h = k$), we augment the side trees with every level-$(k-1)$ node in STR_{k-1} and obtain STR_k. The explicit storage of these side trees requires $O(\log n)$ bits per node. However, the desired storage space of $O(\log \sigma)$ bits per node is achieved via a novel encoding technique. For every level-$(k-1)$ node u in STR_k, we define the followings:

- ℓ_i^u represents the ith leftmost leaf in $Sidetree(u)$
- $E_u[1..n_u]$ be an array of characters, where $E_u[i] = T[pos(\ell_i^u) + |path(u)| + 1]$, and n_u represents the number of leaves in $Sidetree(u)$.
- $B_u[1..\sigma]$ be a bit vector of length σ, where $B_u[z] = 1$ if and only if there exists an outgoing edge from u with $z \in \Sigma$ as the leading character.

The following lemma summarizes the key idea behind our result.

Lemma 3. *For any level-$(k-1)$ node u and $i \in [1, n_u]$, $pos(\ell_i^u)$ is the same as $pos(\cdot)$ of the $prank_{E_u}(i)$th leftmost leaf node in the subtree of node w, where w is a child of u, with $E_u[i]$ the leading character on the edge connecting u and w.*

Proof. Corresponding to every leaf node in the subtree of any child node of u, except the one on the same heavy path as that of u, there exists another unique leaf node in $Sidetree(u)$, such that both have the same $pos(\cdot)$ value. Then the lemma follows from the fact that, the character at the position $|path(u)| + 1$ is the same for the suffix corresponding to any two leaves in the subtree of w, and therefore the lexicographic ordering of those suffixes remains unchanged even after replacing the $(|path(u)| + 1)$th character by ϕ. □

Based on the key observation in the above lemma, we obtain the following result.

Lemma 4. *By maintaining an $O(n \log^k n \log \sigma)$ bits structure, we can compute $pos(\ell_{u_i})$ for any $i \in [1, n_u]$ for any level-$(k-1)$ node u in $O(1)$ time.*

Proof. First we maintain the tree structure of STR_k using succinct data structures [24] in $O(n \log^k n)$ bits of space. Then for every level-$(k-1)$ node u, we maintain E_u and the supporting structures for constant time partial rank queries (refer to Section 2.4) on E_u, in total $O(\sum n_u \log \sigma) = O(n \log^k n \log \sigma)$ bits. Also maintain $B_u[1..\sigma]$ corresponding to all level-$(k-1)$ nodes u, where $B_u[1...\sigma]$ for a particular node u can be maintained in $O(degree(u) \log(\sigma/degree(u)))$ bits or $O(degree(u))$ words of space using an indexible dictionary [23]. Notice that the total space (in words) for maintaing all such bit vectors can be asymptotically bounded by the number of level-$(k-1)$ nodes, which is $O(n \log^{k-1} n)$. By combining the above pieces, the overall space can be bounded by $O(n \log^k n \log \sigma)$ bits. Using these structure, combined with the result in Lemma 3, $pos(\ell_{u_i})$ for any $i \in [1, n_u]$ for any level-$(k-1)$ node u can be answered in $O(1)$ time as follows:

– Find the child node w of u, such that the leading character on the edge connecting u and w is $E_u[i]$ using the following steps: find $k = rank_{B_u}(E_u[i])$ (notice that $B_u[E_u[i]] = 1$, therefore k can be computed in $O(1)$ from B_u, which is maintained using an indexible dictionary) and w is given by the kth leftmost child of u (which can be identified in constant time using the tree structure of STR_k).
– Report $pos(\cdot)$ of the $(prank_{E_u}(i))$th leaf node in the subtree of w, which is again a constant time operation. □

Lemma 5. *The structure STR_k can be encoded in $O(n \log^k n \log \sigma)$ bits such that $pos(\cdot)$ of any of its leaf node can be computed in $O(1)$ time.*

Proof. The $pos(\cdot)$ values corresponding to all those leaves, which are not a *level-k* node can be maintained explicitly in $O(n \log^k n)$ bits. This is because the number of such leaves is $O(n \log^{k-1} n)$. In order to encode these values efficiently for *level-k* leaf nodes, we first mark all those nodes in STR_k which are not *level-k*. The information whether a node in STR_k is marked or not can be maintained using a bit vector B of length equal to the number of nodes in STR_k, where $B[i] = 1$ iff the ith node (in terms of pre-order rank) is marked. Then, $pos(\ell_j)$ of any *level-k* (i.e., unmarked) leaf node can be computed as follows: first find the lowest marked ancestor u of ℓ_j. Let ℓ_j be the ith leftmost leaf in the subtree of u, where $i = j - f + 1$ and ℓ_f is the leftmost leaf in the subtree of u (notice that f can be computed in $O(1)$ time). Therefore $pos(\ell_j) = pos(\ell_i^u)$ and can be decoded in $O(1)$ time using the result of Lemma 4. □

A pattern matching query on our encoded STR_k can be performed in the same standard way. Notice that using an LCP data structure, the locus nodes can be identified in $O(p + 2^h \log\log n)$ time. However its space occupancy is $O(n \log^{k+1} n)$ bits and we cannot afford to maintain it within the desired space complexity. Therefore, (although slower) we use the $O(p + 2^h \log n)$ time algorithm described in Section 4.1 for identifying the locus nodes. As $pos(\cdot)$ for any leaf node can be decoded in $O(1)$ time (refer to Lemma 5), after finding the locus nodes, it takes only $O(occ)$ time to report the occurrences. By combining the above pieces, we have the following final result.

Theorem 4. *A given text T of length n can be indexed in $O(n \log^k n \log \sigma)$ bits, such that all those occurrences of a pattern P containing $h \leq k$ wildcards can be retrieved in $O(p + 2^h \log n + occ)$ time.* □

References

1. Alstrup, S., Brodal, G.S., Rauhe, T.: New data structures for orthogonal range searching. In: FOCS, pp. 198–207 (2000)
2. Amir, A., Keselman, D., Landau, G.M., Lewenstein, M., Lewenstein, N., Rodeh, M.: Text indexing and dictionary matching with one error. J. Algorithms 37(2), 309–325 (2000)
3. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Monotone minimal perfect hashing: searching a sorted table with O(1) accesses. In: SODA, pp. 785–794 (2009)
4. Belazzougui, D., Navarro, G., Valenzuela, D.: Improved compressed indexes for full-text document retrieval. J. Algorithms 18, 3–13 (2013)
5. Bille, P., Gørtz, I.L.: Substring range reporting. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 299–308. Springer, Heidelberg (2011)
6. Bille, P., Gørtz, I.L., Vildhøj, H.W., Vind, S.: String indexing for patterns with wildcards. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 283–294. Springer, Heidelberg (2012)
7. Bucher, P., Bairoch, A.: A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In: ISMB, pp. 53–61 (1994)
8. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: Compressed indexes for approximate string matching. Algorithmica 58(2), 263–281 (2010)
9. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Symposium on Computational Geometry, pp. 1–10 (2011)
10. Chien, Y.-F., Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: Geometric BWT: Compressed text indexing via sparse suffixes and range searching. Algorithmica (2013)

11. Cole, R., Gottlieb, L.-A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: STOC, pp. 91–100 (2004)
12. Golynski, A., Ian Munro, J., Srinivasa Rao, S.: Rank/select operations on large alphabets: a tool for text indexing. In: SODA, pp. 368–373 (2006)
13. Hofmann, K., Bucher, P., Falquet, L., Bairoch, A.: The prosite database, its status in 1999. Nucleic Acids Research 27(1), 215–219 (1999)
14. Hon, W.-K., Ku, T.-H., Shah, R., Thankachan, S.V., Vitter, J.S.: Compressed text indexing with wildcards. J. Discrete Algorithms 19, 23–29 (2013)
15. Huynh, T.N.D., Hon, W.-K., Lam, T.-W., Sung, W.-K.: Approximate string matching using compressed suffix arrays. Theoretical Comp. Science 352(1), 240–249 (2006)
16. Iliopoulos, C.S., Rahman, M.S.: Indexing factors with gaps. Algorithmica 55(1), 60–70 (2009)
17. Kärkkäinen, J., Puglisi, S.J.: Medium-space algorithms for inverse BWT. ESA (1), 451–462 (2010)
18. Lam, T.-W., Sung, W.-K., Tam, S.-L., Yiu, S.-M.: Space efficient indexes for string matching with don't cares. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 846–857. Springer, Heidelberg (2007)
19. Lewenstein, M.: Indexing with gaps. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 135–143. Springer, Heidelberg (2011)
20. Lewenstein, M.: Orthogonal range searching for text indexing. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Ianfest-66. LNCS, vol. 8066, pp. 267–302. Springer, Heidelberg (2013)
21. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
22. Rahman, M.S., Iliopoulos, C.S.: Pattern matching algorithms with don't cares. In: SOFSEM (2), pp. 116–126 (2007)
23. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4) (2007)
24. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: SODA, pp. 134–149 (2010)
25. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. 26(3), 362–391 (1983)
26. Tam, A., Wu, E., Lam, T.-W., Yiu, S.-M.: Succinct text indexing with wildcards. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 39–50. Springer, Heidelberg (2009)
27. Thachuk, C.: Compressed indexes for text with wildcards. Theor. Comput. Sci. 483, 22–35 (2013)
28. Weiner, P.: Linear pattern matching algorithms. In: SWAT (FOCS), pp. 1–11 (1973)

On Determining Deep Holes of Generalized Reed-Solomon Codes *

Qi Cheng[1], Jiyou Li[2], and Jincheng Zhuang[1]

[1] School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
qcheng@cs.ou.edu, jzhuang@ou.edu
[2] Department of Mathematics
Shanghai Jiao Tong University
Shanghai, PR China
lijiyou@sjtu.edu.cn

Abstract. For a linear code, deep holes are defined to be vectors that are further away from codewords than all other vectors. The problem of deciding whether a received word is a deep hole for generalized Reed-Solomon codes is proved to be co-NP-complete [9][5]. For the extended Reed-Solomon codes $RS_q(\mathbb{F}_q, k)$, a conjecture was made to classify deep holes in [5]. Since then a lot of effort has been made to prove the conjecture, or its various forms. In this paper, we classify deep holes completely for generalized Reed-Solomon codes $RS_p(D, k)$, where p is a prime, $|D| > k \geqslant \frac{p-1}{2}$. Our techniques are built on the idea of deep hole trees, and several results concerning the Erdös-Heilbronn conjecture.

Keywords: Reed-Solomon code, deep hole, deep hole tree, Erdös-Heilbronn conjecture.

1 Introduction

Reed-Solomon codes are of special interest and importance both in theory and practice of error-correcting.

Definition 1. *Let $\mathbb{F}_q$ be a finite field with q elements and characteristic p. Let $D = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_q$ be the evaluation set and $v_i \in \mathbb{F}_q^*, 1 \leqslant i \leqslant n$, be the column multipliers. The set of codewords of the generalized Reed-Solomon code $RS_q(D, k)$ of length n and dimension k over $\mathbb{F}_q$ is defined as*

$$RS_q(D, k) = \{(v_1 f(\alpha_1), \ldots, v_n f(\alpha_n)) \in \mathbb{F}_q^n \mid f(x) \in \mathbb{F}_q[x], deg(f) \leqslant k - 1\}.$$

* The research is partially supported by 973 Program (Grant 2013CB834201) and NSF under grants CCF-0830522 and CCF-0830524 for Q.C. and J.Z., and by National Science Foundation of China (11001170) and Ky and Yu-Fen Fan Fund Travel Grant from the AMS for J.L.. The full version appears as arXiv:1309.3546.

We will write generalized Reed-Solomon codes as GRS codes for short in the sequel. If $D = \mathbb{F}_q^*$, it is called *primitive*. If $D = \mathbb{F}_q$, it is called a *singly-extended* GRS code. A GRS code is called *normalized* if its column multipliers are all equal to 1. In this paper, we will work on the normalized GRS without loss of generality. And we will write singly-extended GRS codes as extended GRS codes for short.

The encoding algorithm of the GRS code can be described by the linear map $\varphi : \mathbb{F}_q^k \to \mathbb{F}_q^n$, in which a message $(a_1, \ldots, a_k)$ is mapped to a codeword $(f(\alpha_1), \ldots, f(\alpha_n))$, where $f(x) = a_k x^{k-1} + a_{k-1} x^{k-2} + \cdots + a_1 \in \mathbb{F}_q[x]$.

The *Hamming distance* between two words is the number of their distinct coordinates. The *error distance* of a received word $u \in \mathbb{F}_q^n$ to the code is defined as its minimum Hamming distance to codewords. The *minimum distance* of a code, which is denoted by d, is the smallest distance between any two distinct codewords of the code. The *covering radius* of a code is the maximum distance from any vector in $\mathbb{F}_q^n$ to the nearest codeword. A *deep hole* is a vector achieving the covering radius. A linear code $[k, d]$ is called *maximum distance separable* (in short, MDS) if it attains the Singleton bound, i.e., $k = n - d + 1$. GRS code is a linear MDS code, and its minimum distance is known to be $n - k + 1$ and the covering radius is known to be $n - k$. Thus for the GRS code, u is a deep hole if $d(u, RS_q(D, k)) = n - k$. A linear code can be represented by a generator matrix. In this paper, we assume that the rows of a generator matrix form a basis for the code.

1.1 Related Work

Efforts have been made to obtain an efficient decoding algorithm for GRS codes. Given a received word $u \in \mathbb{F}_q^n$, if the error distance is smaller than $n - \sqrt{nk}$, then the list decoding algorithm of Sudan [17] and Guruswami-Sudan [8] solves the decoding in polynomial time. However, in general, the maximum likelihood decoding of GRS codes is NP-hard [9].

We would like to determine all the deep holes of the code. To this end, given a received word $u = (u_1, u_2, \ldots, u_n) \in \mathbb{F}_q^n$, we consider the following Lagrange interpolating polynomial

$$u(x) = \sum_{i=1}^{n} u_i \frac{\prod_{j \neq i}(x - \alpha_j)}{\prod_{j \neq i}(\alpha_i - \alpha_j)} \in \mathbb{F}_q[x],$$

where $D = \{\alpha_1, \ldots, \alpha_n\}$ is the evaluation set. The Lagrange interpolating polynomial is the only polynomial in $\mathbb{F}_q[x]$ of degree less than n that satisfies $u(\alpha_i) = u_i, 1 \leqslant i \leqslant n$. In this paper, we say that a function $u(x)$ *generates* a vector $u \in \mathbb{F}_q^n$ if $u = (u(\alpha_1), u(\alpha_2), \ldots, u(\alpha_n))$. We have the following conclusions:

– If $deg(u) \leqslant k-1$, then $u \in RS_q(D, k)$ by definition and $d(u, RS_q(D, k)) = 0$.
– If $deg(u) = k$, then it can be shown that u is a deep hole by the following proposition [10], i.e., $d(u, RS_q(D, k)) = n - k$.

Proposition 1. *([10]) For $k \leqslant deg(u) \leqslant n-1$, we have the inequality*

$$n - deg(u) \leqslant d(u, RS_q(D,k)) \leqslant n-k.$$

When the degree of $u(x)$ becomes larger than k, the situation becomes complicated for GRS codes. However, in the case of (singly-)extended GRS codes, the situation seems to be much simpler. Cheng and Murray [5] conjectured in 2007 that the vectors generated by polynomial of degree k are the only possible deep holes.

Conjecture 1. ([5]) A word u is a deep hole of $RS_q(\mathbb{F}_q, k)$ if and only if $\deg(u) = k$.

There is an analogous conjecture for deep holes of primitive Reed-Solomon codes by Wu and Hong [20].

Conjecture 2. ([20]) A word u is a deep hole of $RS_q(\mathbb{F}_q^*, k)$ if and only if:

$$u(x) = ax^k + f_{\leqslant k-1}(x), a \neq 0;$$

or

$$u(x) = bx^{q-2} + f_{\leqslant k-1}(x), b \neq 0;$$

where $f_{\leqslant k-1}(x)$ denotes a polynomial with degree not larger than $k-1$.

Cheng and Murray [5] got the first result by reducing the problem to the existence of rational points on a hypersurface over $\mathbb{F}_q$.

Theorem 1. *[5] Let $u \in \mathbb{F}_q^q$ such that $1 \leqslant \deg(u) - k \leqslant q-1-k$. If $q \geqslant \max(k^{7+\epsilon}, d^{\frac{13}{3}+\epsilon})$ for some constant $\epsilon > 0$, then u is not a deep hole.*

Following a similar approach of Cheng-Wan [6], Li and Wan [12] improved the above result with Weil's character sum estimate.

Theorem 2. *[12] Let $u \in \mathbb{F}_q^q$ such that $1 \leqslant \deg(u) - k \leqslant q-1-k$. If*

$$q > \max((k+1)^2, d^{2+\epsilon}), k > (\frac{2}{\epsilon}+1)d + \frac{8}{\epsilon} + 2$$

for some constant $\epsilon > 0$, then u is not a deep hole.

Then Liao [13] proved the following result:

Theorem 3. *[13] Let $r \geqslant 1$ be an integer. For any received word $u \in \mathbb{F}_q^q, r \leqslant \deg(u) - k \leqslant q-1-k$, if*

$$q > \max(2\binom{k+r}{2} + d, d^{2+\epsilon}), k > (\frac{2}{\epsilon}+1)d + \frac{2r+4}{\epsilon} + 2$$

for some constant $\epsilon > 0$, then $d(u, RS_q(\mathbb{F}_q, k)) \leqslant q-k-r$, which implies that u is not a deep hole.

Antonio Cafure etc. [4] proved the following result with tools of algebraic geometry:

Theorem 4. *[4] Let $u \in \mathbb{F}_q^q$ such that $1 \leqslant \deg(u) - k \leqslant q - 1 - k$. If*

$$q > \max((k+1)^2, 14d^{2+\epsilon}), k > (\frac{2}{\epsilon} + 1)d,$$

for some constant $\epsilon > 0$, then u is not a deep hole.

Using Weil's character sum estimate and Li-Wan's new sieve [11] for distinct coordinates counting, Zhu and Wan [21] showed the following result:

Theorem 5. *[21] Let $r \geqslant 1$ be an integer. For any received word $u \in \mathbb{F}_q^q, r \leqslant \deg(u) - k \leqslant q - 1 - k$, there are positive constants c_1 and c_2 such that if*

$$d < c_1 q^{1/2}, (\frac{d+r}{2} + 1)\log_2(q) < k < c_2 q,$$

then $d(u, RS_q(\mathbb{F}_q, k)) \leqslant q - k - r$.

The deep hole problem for Reed-Solomon codes is also closely related to the famous MDS conjecture in coding theory. On one hand, GRS codes are MDS codes. On the other hand, it is known that all long enough MDS codes are essentially GRS codes. Following the notation of [14], let $N_{min}(k, q)$ be the minimal integer, if any, such that every $[n, k]$ MDS code over $\mathbb{F}_q$ with $n > N_{min}(k, q)$ is GRS and be $q + 2$ if no such integer exists. For the case of $k = 3$, Segre [15] obtained the following result:

Theorem 6. *[15] If q is odd, every $[n, 3]$ MDS code over $\mathbb{F}_q$ with $q - \frac{\sqrt{q} - 7}{4} < n \leqslant q + 1$ is GRS.*

When $q = p$ is a prime, Voloch [18] obtained the following result:

Theorem 7. *[18] If p is an odd prime number, every $[n, 3]$ MDS code over $\mathbb{F}_p$ with $p - \frac{p}{45} + 2 < n \leqslant p + 1$ is GRS.*

Further, there is a relation for $N_{min}(k + 1, q)$ and $N_{min}(k, q)$ [14] as follows:

Lemma 1. *[14] For $3 \leqslant k \leqslant q - 2$, we have*

$$N_{min}(k + 1, q) \leqslant N_{min}(k, q) + 1.$$

Ball [2] showed the following result:

Theorem 8. *[2] Let S be a set of vectors of the vector space $\mathbb{F}_q^k$, with the property that every subset of S of size k is a basis. If $|S| = q + 1$ and $k \leqslant p$ or $3 \leqslant q - p + 1 \leqslant k \leqslant q - 2$, where p is the characteristic of $\mathbb{F}_q$, then S is equivalent to the following set:*

$$\{(1, \alpha, \alpha^2, \dots, \alpha^{k-1}) \mid \alpha \in \mathbb{F}_q\} \cup \{(0, \dots, 0, 1)\}.$$

1.2 Our Result

In this paper, we classify the deep holes in many cases. Firstly, we show:

Theorem 9. *Let $p > 2$ be a prime number, $k \geqslant \frac{p-1}{2}$, $D = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ with $k < n \leqslant p$. The only deep holes of $RS_p(D,k)$ are generated by functions which are equivalent to the following:*

$$f(x) = x^k, \quad f_\delta(x) = \frac{1}{x-\delta},$$

where $\delta \in \mathbb{F}_p \setminus D$. Here two functions $f(x)$ and $g(x)$ are equivalent if and only if there exists $a \in \mathbb{F}_p^*$ and $h(x) \in \mathbb{F}_p[x]$ with degree less than k such that

$$g(x) = af(x) + h(x).$$

Our techniques are built on the idea of deep hole trees, and several results concerning the Erdös-Heilbronn conjecture. We also show the following theorem based on some results of finite geometry.

Theorem 10. *Given a finite filed $\mathbb{F}_q$ with characteristic $p > 2$, we have*

- *If $k+1 \leqslant p$ or $3 \leqslant q-p+1 \leqslant k+1 \leqslant q-2$, then Conjecture 1 is true.*
- *If $3 \leqslant k < \dfrac{\sqrt{q}+1}{4}$, then Conjecture 2 is true.*
- *If $3 \leqslant k < \dfrac{p}{45}$, where $q = p$ is prime, then Conjecture 2 is true.*

This paper is organized as follows: Section 2 presents some preliminaries; Section 3 describes the idea of the deep hole tree; Section 4 demonstrates the proof of Theorem 9.

2 Preliminaries

2.1 A Criterion for Deep Holes of Linear MDS Codes

By definition, deep holes of a linear code are words that has a maximum distance to the code. In the case of linear MDS codes, there is another way to characterize the deep hole as follows, which connects the concept of deep holes with the MDS codes. The following is well known:

Proposition 2. *Let $\mathbb{F}_q$ be a finite field with characteristic p. Suppose G is a generator matrix for $RS_q(D,k)$ with covering radius $\rho = |D| - k = n - k$, then $u \in \mathbb{F}_q^n$ is a deep hole of $RS_q(D,k)$ if and only if*

$$G' = \left[\frac{G}{u}\right]$$

generates another linear maximum distance separable code.

2.2 Some Additive Combinatorics Results

In this section, we introduce some additive combinatorics results that we will use later. The first theorem is about the estimation of the size of restricted sum sets, which is first proved by Dias da Silva and Hamidoune [16]. Then Alon etc. [1] gave a simple proof using the polynomial method.

Theorem 11. *[16,1] Let $\mathbb{F}$ be a field with characteristic p and n be a positive integer. Then for any finite subset $S \subset \mathbb{F}$ we have*

$$|n^\wedge S| \geqslant \min\{p, n|S| - n^2 + 1\},$$

where $n^\wedge S$ denotes the set of all sums of n distinct elements of S.

Brakemeier [3] and Gallardo etc. [7] established the following theorem:

Theorem 12. *[3,7] Let n be a positive integer and $S \subset \mathbb{Z}/n\mathbb{Z}$. If $|S| > \frac{n}{2} + 1$, then*

$$2^\wedge S = \mathbb{Z}/n\mathbb{Z},$$

where $2^\wedge S$ denotes the set of all sums of 2 distinct elements of S.

Hence we have the following corollary:

Corollary 1. *Let $\mathbb{F}_p$ be a prime finite field, $S \subset \mathbb{F}_p^*$. If $|S| > \frac{p+1}{2}$, then each element of $\mathbb{F}_p^*$ is the product of two distinct elements of S.*

3 Construction of the Deep Hole Tree

Let $\mathbb{F}_q = \{\alpha_1, \alpha_2, \cdots, \alpha_q = 0\}$. The polynomials in $\mathbb{F}_q[x]$ of degree less than q forms a $\mathbb{F}_q$-linear space, with a basis

$$\{1, x, \ldots, x^{k-1}, \prod_{i=1}^{k}(x - \alpha_i), \ldots, \prod_{i=1}^{q-1}(x - \alpha_i)\}.$$

Given a polynomial $f(x) \in \mathbb{F}_q[x]$ with degree $q - 1$ we have

$$f(x) = l(x) + c_1 \prod_{i=1}^{k}(x - \alpha_i) + \cdots + c_{q-k} \prod_{i=1}^{q-1}(x - \alpha_i),$$

where $l(x)$ is of degree less than k, we want to determine when $f(x)$ generates a deep hole of $RS_q(\mathbb{F}_q, k)$. By Proposition 2, $f(x)$ generates a deep hole if and only if

$$G' = \begin{bmatrix} G \\ \hline u \end{bmatrix}$$

generates an MDS code, where G is the generator matrix of $RS_q(\mathbb{F}_q, k)$, and $u = (f(\alpha_1), \ldots, f(\alpha_q))$.

Observe that the function, which generates a deep hole for $RS_q(D_2, k)$, also generates a deep hole for $RS_q(D_1, k)$ if $D_1 \subset D_2$. Instead of considering the deep holes for $RS_q(\mathbb{F}_q, k)$ at the first step, we propose to consider a smaller evaluation set at the beginning and make it increase gradually. To be more precise, firstly we determine c_1 over $D_1 = \{\alpha_1, \ldots, \alpha_{k+1}\}$, then we determine c_2 over $D_2 = \{\alpha_1, \ldots, \alpha_{k+2}\}$ based on the knowledge of c_1, so on and so forth. We present the result as a tree, which we will call a *deep hole tree* in the sequel.

Remark 1. Wu and Hong [19] showed that if $D = \mathbb{F}_q \setminus \{\beta_1, \ldots, \beta_l\}$ then $f_{\beta_i}(x) = \frac{1}{x-\beta_i}$ generates a deep hole for $RS_q(D, k)$, where $1 \leqslant i \leqslant l$. We can also deduce this from Proposition 2. For convenience, we will call these deep holes and deep holes generated by a function of degree k *expected deep holes.*

Motivated by Remark 1, firstly we construct the *expected deep hole tree* as follows:

- The root node is 1 without loss of generality, i.e., $c_1 = 1$.
- There are $p-k-1$ branches of the tree, each with distinct length in $[2, p-k]$. And we designate the sequence of nodes in a branch with length l as b_l.
  - If $l = p - k$, then $b_{p-k} = (0, \ldots, 0)$.
  - If $2 \leqslant l \leqslant p-k-1$, then $b_l = (c_1, \ldots, c_l)$, where $f = \frac{1}{x-\alpha_{l+1}}$ is equivalent to $c_1 \prod_{i=1}^{k}(x - \alpha_i) + \cdots + c_l \prod_{i=1}^{k+l-1}(x - \alpha_i)$.

Proposition 3. *The expected deep hole tree is a part of the full deep hole tree.*

Proof. This follows from Remark 1.

Now we can construct the full deep hole tree based on the expected deep hole tree.

- The root node is 1 without loss of generality, i.e., $c_1 = 1$.
- The children $\{c_{i+1}\}$ of a node $c_i, 1 \leqslant i \leqslant q-k-1$ are defined as follows: given the ancestors $(c_1, \ldots, c_i)$, for $\gamma \in \mathbb{F}_q$, if γ is the child of c_i in the expected deep hole tree, then keep it; otherwise, if

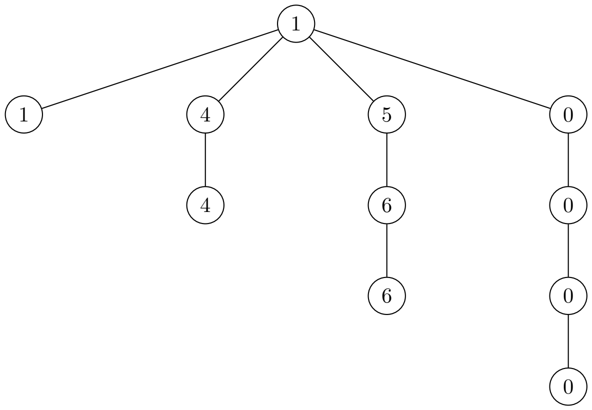
$$c_1 \prod_{i=1}^{k}(x - \alpha_i) + \cdots + c_i \prod_{i=1}^{k+i-1}(x - \alpha_i) + \gamma \prod_{i=1}^{k+i}(x - \alpha_i)$$

  satisfies the property of the function which generates a deep hole as in Proposition 2, then γ is a child of c_i.

That is, we keep the nodes of the expected deep hole tree and add additional ones if necessary. Now we illustrate the procedure to construct the deep hole tree by one example.

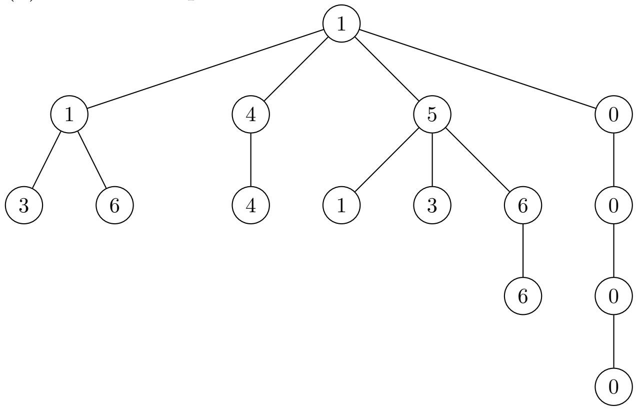
Example 1. Let $p = 7, k = 2$. The evaluation set is ordered such that $\alpha_i = i, 1 \leqslant i \leqslant 7$.

(1) The expected deep hole tree is as follows:



The root is corresponding to the evaluation set $D_1 = \{1, 2, 3\}$. The expected deep holes are generated by functions equivalent to $\prod_{i=1}^{2}(x-i)$. In depth 2, the evaluation set is $D_2 = \{1, 2, 3, 4\}$. One of the expected deep holes is generated by the function $\prod_{i=1}^{2}(x-i) + \prod_{i=1}^{3}(x-i)$, which is equivalent to $f_5 = \frac{1}{x-5}$. In depth 3, the evaluation set is $D_3 = \{1, 2, 3, 4, 5\}$. One of the expected deep holes is generated by the function $\prod_{i=1}^{2}(x-i) + 4\prod_{i=1}^{3}(x-i) + 4\prod_{i=1}^{4}(x-i)$, which is equivalent to $f_6 = \frac{1}{x-6}$. In depth 4, the evaluation set is $D_4 = \{1, 2, 3, 4, 5, 6\}$. One of the expected deep holes is generated by the function $\prod_{i=1}^{2}(x-i) + 5\prod_{i=1}^{3}(x-i) + 6\prod_{i=1}^{4}(x-i) + 6\prod_{i=1}^{5}(x-i)$, which is equivalent to $f_0 = \frac{1}{x}$. In depth 5, the evaluation set is $D_5 = \{1, 2, 3, 4, 5, 6, 7\}$. One of the expected deep holes is generated by the function $\prod_{i=1}^{2}(x-i)$.

(2) The full deep hole tree is as follows:



Note that there are more nodes here than the expected ones. For example, in depth 3, there is an additional deep hole generated by the function $\prod_{i=1}^{2}(x-i) + \prod_{i=1}^{3}(x-i) + 3\prod_{i=1}^{4}(x-i)$. Also, there is an additional deep hole generated by the function $\prod_{i=1}^{2}(x-i) + 5\prod_{i=1}^{3}(x-i) + \prod_{i=1}^{4}(x-i)$.

4 Proof of Theorem 9

The basic idea of the proof of Theorem 9 is reducing the problem to some additive number theory problems. We first present several lemmas.

Lemma 2. *In depth $d = 2$, the nodes are the same in both the expected deep hole tree and full deep hole tree.*

Lemma 3. *Let p be an odd prime, $k \geqslant \frac{p-1}{2}, d \geqslant 2$ be a positive integer and $D_d = \{\alpha_1, \ldots, \alpha_{k+d}\} \subset \mathbb{F}_p, \delta \in \mathbb{F}_p \setminus D_d$. For any $\gamma \in \mathbb{F}_p$, there exists a subset $\{\beta_1, \ldots, \beta_k\} \subset D_d$ such that the matrix*

$$A = \begin{bmatrix} 1 & \cdots & 1 & 1 \\ \beta_1 & \cdots & \beta_k & \delta \\ \vdots & \ddots & \vdots & \vdots \\ \beta_1^{k-1} & \cdots & \beta_k^{k-1} & \delta^{k-1} \\ \frac{1}{\beta_1-\delta} & \cdots & \frac{1}{\beta_k-\delta} & \gamma \end{bmatrix}$$

is singular.

Lemma 4. *Let p be an odd prime, $k \geqslant \frac{p-1}{2}, d \geqslant 2$ be a positive integer and $D_{d+1} = \{\alpha_1, \ldots, \alpha_{k+d+1} = \delta\} \subset \mathbb{F}_p$. For any $\delta' \in \mathbb{F}_p, \delta' \notin D_{d+1}, \gamma \in \mathbb{F}_p, \gamma \neq \frac{1}{\delta-\delta'}$, there exists a subset $\{\beta_1, \ldots, \beta_k\} \subset D_{d+1} \setminus \{\delta\}$ such that the matrix*

$$B = \begin{bmatrix} 1 & \cdots & 1 & 1 \\ \beta_1 & \cdots & \beta_k & \delta \\ \vdots & \ddots & \vdots & \vdots \\ \beta_1^{k-1} & \cdots & \beta_k^{k-1} & \delta^{k-1} \\ \frac{1}{\beta_1-\delta'} & \cdots & \frac{1}{\beta_k-\delta'} & \gamma \end{bmatrix}$$

is singular.

Now we prove Theorem 9.

Proof. (of Theorem 9) Proceed by induction on the depth of the full deep hole tree.
Basis case This follows from Lemma 2.
Inductive step We need to show that if the set of nodes of the full deep hole tree coincide with the nodes of the expected deep hole tree in the same depth $d \geqslant 2$, then there are no additional nodes in depth $d + 1$ except the expected ones. Denote the corresponding evaluation set by $D_d = \{\alpha_1, \ldots, \alpha_{k+d}\}$ in depth d and $D_{d+1} = \{\alpha_1, \ldots, \alpha_{k+d}, \alpha_{k+d+1} = \delta\}$ in depth $d+1$. In order to show there are no new nodes in depth $d + 1$, There are two cases to consider.
Case 1: We need to show the branch, which is corresponding to the function $f = \frac{1}{x-\delta}$, will not continue in the depth $d+1$. It suffices to show that there exists

a subset $\{\beta_1, \dots, \beta_k\} \subset \{\alpha_1, \dots, \alpha_{k+d}\}$ such that for any $\gamma \in \mathbb{F}_p$ and matrix

$$A = \begin{bmatrix} 1 & \cdots & 1 & 1 \\ \beta_1 & \cdots & \beta_k & \delta \\ \vdots & \ddots & \vdots & \vdots \\ \beta_1^{k-1} & \cdots & \beta_k^{k-1} & \delta^{k-1} \\ \frac{1}{\beta_1-\delta} & \cdots & \frac{1}{\beta_k-\delta} & \gamma \end{bmatrix},$$

we have $\det(A) = 0$. This follows from Lemma 3.
Case 2: We need to show that the branch, which is corresponding to the function $f = \frac{1}{x-\delta'}$, where $\delta' \notin D_{k+1}$, has only one child in depth $k+1$. It suffices to show that there exists a subset $\{\beta_1, \dots, \beta_k\} \subset D_d$ such that for any $\delta' \notin D_{d+1}, \gamma \in \mathbb{F}_p, \gamma \neq \frac{1}{\delta-\delta'}$ and matrix

$$B = \begin{bmatrix} 1 & \cdots & 1 & 1 \\ \beta_1 & \cdots & \beta_k & \delta \\ \vdots & \ddots & \vdots & \vdots \\ \beta_1^{k-1} & \cdots & \beta_k^{k-1} & \delta^{k-1} \\ \frac{1}{\beta_1-\delta'} & \cdots & \frac{1}{\beta_k-\delta'} & \gamma \end{bmatrix},$$

we have $\det(B) = 0$. This follows from Lemma 4.

From the principle of induction, the theorem is proved.

5 Concluding Remarks

In this paper, we classify deep holes completely for Generalized Reed-Solomon codes $RS_p(D, k)$, where p is a prime, $|D| > k \geqslant \frac{p-1}{2}$. We suspect that a similar result hold over finite fields of composite order, and leave it as an open problem.

Acknowledgement. The authors would like to thank the anonymous referees for helpful comments.

References

1. Nathanson, M., Alon, A., Ruzsa, I.: The polynomial method and restricted sums of congruence classes. Journal of Number Theory 56(2), 404–417 (1996)
2. Ball, S.: On sets of vectors of a finite vector space in which every subset of basis size is a basis. Journal of the European Mathematical Society 14(3), 733–748 (2012)
3. Brakemeier, W.: Eine anzahlformel von zahlen modulo n. Monatshefte für Mathematik 85, 277–282 (1978)
4. Cafure, A., Matera, G., Privitelli, M.: Singularities of symmetric hypersurfaces and an application to reed-solomon codes. Advances in Mathematics of Communications 6(1), 69–94 (2012)

5. Cheng, Q., Murray, E.: On deciding deep holes of reed-solomon codes. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 296–305. Springer, Heidelberg (2007)
6. Cheng, Q., Wan, D.: On the list and bounded distance decodability of Reed-Solomon codes. SIAM Journal on Computing 37(1), 195–209 (2007)
7. Gallardo, L., Grekos, G., Pihko, J.: On a variant of the Erdös-Ginzburg-Ziv theorem. Acta Arithmetica 89, 331–336 (1999)
8. Guruswami, V., Sudan, M.: Improved decoding of Reed-Solomon and algebraic-geometry codes. IEEE Transaction on Information Theory 45(6), 1757–1767 (1999)
9. Guruswami, V., Vardy, A.: Maximum-likelihood decoding of Reed-Solomon codes is NP-hard. In: Proceeding of SODA (2005)
10. Li, J., Wan, D.: On the subset sum problem over finite fields. Finite Fields and Their Applications 14, 911–929 (2008)
11. Li, J., Wan, D.: A new sieve for distinct coordinate counting. Science China Mathematics 53(9), 2351–2362 (2010)
12. Li, Y., Wan, D.: On error distance of Reed-Solomon codes. Science in China Series A: Mathematics 51, 1982–1988 (2008)
13. Liao, Q.: On Reed-Solomon codes. Chinese Annals of Mathematics, Series B 32B, 89–98 (2011)
14. Roth, R.M., Lempel, A.: On MDS codes via cauchy matrices. IEEE Transactions on Information Theory 35, 1314–1319 (1989)
15. Segre, B.: Introduction to Galois geometries. Atti dell'Accademia Nazionale dei Lincei Memorie 8, 133–236 (1967)
16. Dias Da Silva, J.A., Hamidoune, Y.O.: Cyclic spaces for grassmann derivatives and additive theory. Bulletin of the London Mathematical Society 26, 140–146 (1994)
17. Sudan, M.: Decoding of Reed-Solomon codes beyond the error-correction bound. Journal of Complexity 13, 180–193 (1997)
18. Voloch, J.F.: Arcs in projective plans over prime fields. Journal of Geometry 38, 198–200 (1990)
19. Wu, R., Hong, S.: On deep holes of generalized Reed-Solomon codes. CoRR, abs/1205.7016 (2012)
20. Wu, R., Hong, S.: On deep holes of standard Reed-Solomon codes. Science China Mathematics 55(12), 2447–2455 (2012)
21. Zhu, G., Wan, D.: Computing error distance of reed-solomon codes. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 214–224. Springer, Heidelberg (2012)

Isomorphism on Subgraph-Closed Graph Classes: A Complexity Dichotomy and Intermediate Graph Classes

Yota Otachi[1] and Pascal Schweitzer[2]

[1] Japan Advanced Institute of Science and Technology, School of Information Science
otachi@jaist.ac.jp
[2] Forschungsinstitut für Mathematik
Eidgenössische Technische Hochschule Zürich
pascal.schweitzer@fim.math.ethz.ch

Abstract We study the graph isomorphism problem for graph classes defined by sets of forbidden subgraphs. We show that there is a complexity dichotomy in case the set of forbidden subgraphs is finite. More precisely, we show that the problem is polynomial-time solvable if the forbidden set contains a forest of subdivided stars and is graph isomorphism complete otherwise. We also show that, assuming that the graph isomorphism problem is not polynomial-time solvable in general, there is no such dichotomy for the cases of infinite sets of forbidden subgraphs. To this end, we conditionally show that there exists a graph class closed under taking subgraphs with intermediate isomorphism problem, i.e., a class on which the isomorphism problem is neither polynomial-time solvable nor graph isomorphism complete.

1 Introduction

The graph isomorphism problem is the algorithmic task to decide whether two given graphs are isomorphic, i.e., whether there exists a bijection from the vertices of one graph to the vertices of the other graph preserving adjacency and non-adjacency. Although the problem is in NP, up to this point it is neither known whether the problem is polynomial-time solvable nor whether it is NP-hard. We refer the reader to [1,7,13] for introductions to the diverse complexity-theoretic results related the isomorphism problem.

In this paper, we are interested in the complexity of the isomorphism problem on graph classes characterized by sets of forbidden (not necessarily induced) subgraphs. These graph classes are exactly the classes closed under taking subgraphs, or, in other words, graph classes closed under vertex and edge deletion.

Related Work. There has been extensive study of the isomorphism problem on graph classes. Ponomarenko [12] showed that the graph isomorphism problem can be solved in polynomial time on any non-trivial, minor-closed class. In other words, graph isomorphism is polynomial-time solvable on any graph class characterized by a non-empty set of forbidden minors. In his monograph, Grohe (see [4]) showed that for every such graph class, a sufficiently high-dimensional Weisfeiler-Lehman algorithm correctly decides isomorphism. Recently, Grohe and Marx [5] developed a structure theory for graphs characterized by forbidden topological minors. They use this theory to show that the graph isomorphism problem can be solved in polynomial time on any graph class defined by a non-empty set of forbidden topological minors.

All of the results mentioned above are concerned with graph classes that are closed under taking subgraphs. Hereditary graph classes, which are graph classes closed under taking induced subgraphs, do not always have this property. Examples of such classes are several forms of intersection graphs, for which various results exists. For interval graphs for example, isomorphism can be solved in polynomial time [10]. More generally, for various intersection models, isomorphism completeness or polynomial-time solvability is known, while for others the complexity remains open (see [3,6,14] for pointers).

For graph classes characterized by one forbidden induced subgraph, Colbourn and Colbourn (see [2, §4.7]) show the following dichotomy: if the forbidden graph is an induced subgraph of P_4, the path on four vertices, then the problem is polynomial-time solvable; otherwise it is GI-complete. Not as much is known for two or more forbidden induced subgraphs. In [8] the complexity of certain classes characterized by two forbidden induced subgraphs is analyzed, where the situation already appears to be more complex.

Our Results. We show that there is a dichotomy for the graph isomorphism problem on graph classes characterized by a finite set of forbidden subgraphs, and that there is no such dichotomy for graph classes characterized by an infinite set of forbidden subgraphs. More precisely, we first show that the problem is polynomial-time solvable if the forbidden set contains a forest of subdivided stars and is GI-complete for any other finite set of forbidden subgraphs. We then show that, assuming the graph isomorphism problem is not in P, there is a graph class defined by an infinite set of forbidden subgraphs for which the problem is neither GI-complete nor in P. Since classes closed under taking subgraphs are in particular closed under taking induced subgraphs, this also proves the conditional existence of intermediate hereditary graph classes.

2 Preliminaries

In this paper we consider exclusively finite graphs. Let G be a graph. By $N_G(v)$ we denote the neighborhood of a vertex v in G. By the *contraction of an edge* $e = \{v, w\} \in E(G)$, we obtain the graph with the vertex set $V(G) \cup \{u_e\} \setminus \{v, w\}$ and the edge set $E(G) \cup \{\{u_e, x\} \mid x \in N_G(v) \cup N_G(w)\} \setminus (E_v \cup E_w)$, where u_e is a new vertex and E_v and E_w are the edges incident to v and w, respectively. By the *vertex dissolution* of a degree-2 vertex v with the two neighbors u and w, we obtain the graph with the vertex set $V(G) \setminus \{v\}$ and the edge set $E(G) \cup \{u, w\} \setminus \{\{u, v\}, \{v, w\}\}$. By the definitions, if v has degree 2, then edge contraction of $\{v, w\}$ is equivalent to the vertex dissolution of v, up to isomorphism.

A graph H is an *induced subgraph* of G if H can be obtained from G by a sequence of vertex deletions. If H is obtained by additionally deleting edges then H is a *subgraph* of G. A graph H is a *topological minor* of G if H can be obtained from a subgraph of G by a sequence of vertex dissolutions. If we use edge contractions instead of vertex dissolutions, then H is a *minor* of G. A graph G *contains* a graph H as an induced subgraph (a subgraph, a topological minor, a minor) if an induced subgraph (a subgraph, a topological minor, a minor, respectively) of G is isomorphic to H.

For a set of graphs $\mathcal{H}$, the *$\mathcal{H}$-minor-free graphs* form the class of all the graphs that contain no graph in $\mathcal{H}$ as a minor. If $\mathcal{H} = \{H\}$, then we write *H-minor-free graphs.* Similarly, we define *$\mathcal{H}$-topological-minor-free graphs*, *$\mathcal{H}$-subgraph-free graphs*, and *$\mathcal{H}$-induced-subgraph-free graphs.* From the definitions, it follows directly that $\mathcal{H}$-minor-free graphs $\subseteq$ $\mathcal{H}$-topological-minor-free graphs $\subseteq$ $\mathcal{H}$-subgraph-free graphs $\subseteq$ $\mathcal{H}$-induced-subgraph-free graphs for any set of graphs $\mathcal{H}$.

The *edge subdivision* on an edge replaces the edge with a path of length 2, making an edge subdivision the reverse operation of a vertex dissolution. A graph is a *subdivision* of a graph H if it can be obtained from H by a sequence of edge subdivisions. With this notation, we see that G contains H as a topological minor if an only if a subgraph of G is isomorphic to a subdivision of H.

A *star* is a graph isomorphic to the complete bipartite graph $K_{1,s}$ for some positive integer s. A *subdivided star* is a subdivision of a star. Note that any subdivision of a subdivided star is also a subdivided star. Since any subdivided star is a tree, we call the disjoint union of subdivided stars a *forest of subdivided stars.*

3 A Dichotomy for Finite Sets of Forbidden Subgraphs

In this section we present a complexity dichotomy of the graph isomorphism problem for $\mathcal{H}$-subgraph-free graphs when $\mathcal{H}$ is finite. In this dichotomy, the presence of a forest of subdivided stars is the deciding factor. This is due to the following property of these graphs.

Lemma 1. *Let H be a forest of subdivided stars. A graph G contains H as a subgraph if and only if it contains H as a topological minor.*

Proof. The "only-if" part follows since every subgraph is also a topological minor. We now prove the "if" part. Observe that every graph obtained by subdividing an edge in H is isomorphic to a graph obtained from H by adding a new vertex adjacent to a vertex that was previously a leaf. From this observation, we see that any subdivision of H is isomorphic to the graph obtained by attaching paths of suitable lengths to leaves. Consequently, every subdivision of H contains H as a subgraph. Therefore, if G contains a subgraph H' isomorphic to a subdivision of H, then this subgraph H', and thus also G, contain a subgraph isomorphic to H. □

Using the lemma, we obtain our complexity dichotomy.

Theorem 1. *If $\mathcal{H}$ is a finite family of graphs, then the graph isomorphism problem for $\mathcal{H}$-subgraph-free graphs is*

1. *polynomial-time solvable if $\mathcal{H}$ contains a forest of subdivided stars, and*
2. *GI-complete otherwise.*

Proof. (1) Let $H \in \mathcal{H}$ be a forest of subdivided stars. By Lemma 1, all $\mathcal{H}$-free graphs are H-topological-minor-free. Therefore, by a result of Grohe and Marx [5], isomorphism of $\mathcal{H}$-free graphs can be decided in polynomial time.

(2) Suppose now that no graph in $\mathcal{H}$ is a forest of subdivided stars. This implies that each $H \in \mathcal{H}$ contains a component with either a cycle or a path between two vertices of degree at least 3. Let ℓ_H be the maximum length among those cycles and paths in H, and let $\ell_{\mathcal{H}} = \max\{\ell_H \mid H \in \mathcal{H}\}$. Note that $\ell_{\mathcal{H}}$ is a fixed constant since $\mathcal{H}$ is a fixed finite set. Now let $\mathcal{G}$ be the class of all graphs and let $\mathcal{G}'$ be the class of the graphs obtained from each graph in $\mathcal{G}$ by subdividing each edge $\ell_{\mathcal{H}}$ times. In each graph in $\mathcal{G}'$, all cycles and all paths connecting vertices of degree at least 3 have lengths more than ℓ_H. Thus there is no graph in $\mathcal{G}'$ that contains a graph in $\mathcal{H}$ as a subgraph. Observe that two graphs are isomorphic if and only if the graphs obtained by subdividing each edge $\ell_{\mathcal{H}}$ times are isomorphic (see [2]). Hence the isomorphism problem is GI-complete for graphs in $\mathcal{G}'$. $\square$

Using the same arguments as in the proof, we can show that the property of forests of subdivided stars proven in Lemma 1 in fact characterize them.

Corollary 1. *Forests of subdivided stars are exactly the graphs H that satisfy the following property: For every graph G, the graph G contains H as a subgraph if and only if it contains G as a topological minor of G.*

Proof. Assuming first that H is a forest of subdivided stars, the claim of the corollary follows from Lemma 1.

We thus assume that H is not a forest of subdivided stars. As in the proof of the previous theorem let ℓ_H be the maximum length among all cycles and paths connecting vertices of degree at least 3 in H. Let G be the graph obtained from H by subdividing every edge ℓ_H times. By construction G contains H as a topological minor, but it does not contain H as a subgraph since G does not contain cycles or paths connecting vertices of degree at least 3 of length ℓ_H. $\square$

4 Isomorphism on Graph Classes, Isomorphism-Completeness and Reduction Types

To prove the existence of an intermediate graph class we need to clarify several definitions that are typically used slightly vaguely. In other situations, for example in the results of the previous section, the nuances of the various definitions make no difference to the argumentation or the conclusion. However, a priori, in our next section they might.

Given an algorithm A that halts within polynomial time with output Yes or No, we can consider the input pairs of labeled graphs (G, H) for which the algorithm correctly decides isomorphism. We say algorithm A *decides isomorphism for a graph class* $\mathcal{C}$ if for all $G, H, \in \mathcal{C}$ and all labeled versions G^{ℓ}, H^{ℓ} the algorithm correctly decides isomorphism of (G^{ℓ}, H^{ℓ}). Note that we require that A always halts within polynomial time, independent of the input graphs being in $\mathcal{C}$. This is no restriction since we can always alter an algorithm to halt in polynomial time, for example by using a yardstick that keeps track of execution time (see [11]).

In our context, a many-one reduction R is a polynomial time algorithm that transforms every pair of labeled graphs (G^{ℓ}, H^{ℓ}) to some pair of labeled graphs $(G'^{\ell'}, H'^{\ell'})$. We define $\text{Red}(R)$ to be the set of pairs of labeled graphs that are the output of the algorithm for some input pair of labeled graphs. The set of unlabeled graphs that appear in some labeled version in $\text{Red}(R)$ is *graph isomorphism complete*. We emphasize that the definition slightly differs from other commonly used definitions, which define a decision problem to be graph isomorphism complete if it is polynomially equivalent to graph isomorphism (see [2]). The difference is that we do not specify what the outcome of the algorithm has to be when one of the input graphs is not within the graph class. This difference becomes irrelevant if the graph class is recognizable in polynomial time, a property we do not know to be true for the intermediate graph class constructed in the next section. On a related note we remark that, while for many graph classes recognition algorithms and isomorphism were developed simultaneously, is it not clear that for arbitrary graph classes the two problems are related in a complexity sense.

The conditional existence of an intermediate graph class proven in the next section also holds for Turing reductions. For those, $\text{Red}(R)$ has to be defined as the set of graphs that appear in some oracle call. The proof of the theorem remains the same. Note that there is no consensus what reduction type should be used in the isomorphism context (see [2]).

Concerning encodings of graphs as bit strings, we will not further specify how exactly labeled graphs are to be encoded. For concreteness we can assume they are given as an adjacency matrix. However, recalling that the reasonable prevalent graph encodings are polynomial-time equivalent, which specific encoding is used is irrelevant for us.

Whatever the specific combination of variants of the definitions may be, a central conclusion common to all of them is that if a graph class is isomorphism complete and graph isomorphism is not polynomial-time solvable then the isomorphism problem on the graph class is also not polynomial-time solvable.

5 An Intermediate Graph Class

Considering infinite sets of forbidden subgraphs, in this section we prove the existence of an intermediate graph class, assuming the general isomorphism problem is not polynomial-time solvable. The idea of the proof follows Ladner's [9] strategy to prove the existence of an NP-intermediate problem (see also [11]). There are some differences though. On the one hand, simplifying our task, we are not concerned with the recognition problem of the intermediate graph class being in NP. On the other hand however, we have to deal with the fact that the sought-after graph class contains unlabeled graphs, while isomorphism algorithms work on pairs of labeled graphs.

Theorem 2. *If graph isomorphism is not solvable in polynomial time then there exists a graph class closed under taking subgraphs on which the isomorphism problem is intermediate, that is, it is neither polynomial-time solvable nor graph isomorphism complete.*

Proof. We assume that graph isomorphism is not solvable in polynomial time and construct a graph class $\mathcal{C}$ that is intermediate. Let $A_1, A_2, \dots$ be a list of all isomorphism algorithms that have a polynomial bound on their running time and always output Yes or No. Let $R_1, R_2, \dots$ be a list of polynomial-time graph isomorphism reductions. Since they comprise subclasses of all algorithms, both the list of algorithms and the list of reductions are countable.

We construct the class $\mathcal{C}$ by alternatingly adding graphs to $\mathcal{C}$ and forbidding a set of graphs that are not to be added to $\mathcal{C}$. To this end, we construct sets of graphs $\mathcal{C}_0, \mathcal{D}_0, \mathcal{C}_1, \mathcal{D}_1, \dots$, such that $\mathcal{C} = \bigcup_{i\in\mathbb{N}} \mathcal{C}_i$, the sets $\mathcal{C}_i$ are all finite, and the forbidden graphs collected in the sets $\mathcal{D}_i$ have bounded girth but are not forests.

The sets $\mathcal{C}_0$ and $\mathcal{D}_0$ are both the empty set. We explain the first step of the construction for $\mathcal{C}_1$ and $\mathcal{D}_1$ and then the inductive construction in general. Since we assume graph isomorphism is not polynomial-time solvable, there is a pair of labeled graphs (G_1^ℓ, H_1^ℓ) that cannot be distinguished by A_1. We add the unlabeled graphs G_1 and H_1 to the graph class which we are in the process of constructing. Since our final class $\mathcal{C}$ is supposed to be closed under taking subgraphs, we also add all subgraphs of G_1 and H_1 to $\mathcal{C}$. Thus, if we denote by $\mathrm{Sub}(G)$ the class of (not necessarily proper) subgraphs of a graph G then we define $\mathcal{C}_1 = \mathrm{Sub}(G_1) \cup \mathrm{Sub}(H_1)$.

Since graph isomorphism is not polynomial-time solvable and since isomorphism of forests is polynomial-time solvable, the set $\mathrm{Red}(R_1)$ (i.e., the set of graph pairs to which R_1 reduces to) contains infinitely many non-forest pairs. In particular it contains a pair of graphs $(G_1'^\ell, H_1'^\ell)$ both not in $\mathcal{C}_1$ such that neither G_1' nor H_1' is a forest. We want to forbid all supergraphs of G_1' and H_1' to be added to our final set $\mathcal{C}$. To prevent this, we add all supergraphs of G_1' and H_1' to $\mathcal{D}_1$. We thus set $\mathcal{D}_1 = \mathrm{Sup}(G_1') \cup \mathrm{Sup}(H_1')$, where for a graph G, $\mathrm{Sup}(G)$ denotes the class of all (not necessarily proper) supergraphs of G.

Inductively we proceed as follows: For $i \in \mathbb{N}$, given a finite set of graphs $\mathcal{C}_i$ that must be included in $\mathcal{C}$ and a set of graphs $\mathcal{D}_i$ of non-forests of bounded girth that are forbidden to be included in $\mathcal{C}$, we obtain $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$ and $\mathcal{D}_i$ as follows: Since for any k, the set of graphs of girth at least k is graph isomorphism complete, there is a pair of graphs $(G_{i+1}^\ell, H_{i+1}^\ell)$ not in $\mathcal{D}_i$ that cannot be distinguished by the algorithm A_{i+1}. We add G_{i+1} and H_{i+1} and all their subgraphs to $\mathcal{C}_i$, i.e., we define $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \mathrm{Sub}(G_i) \cup \mathrm{Sub}(H_i)$. Next we obtain $\mathcal{D}_{i+1}$ from $\mathcal{D}_i$ and $\mathcal{C}_i$ as follows: Consider the set $\mathrm{Red}(R_{i+1})$ to which R_{i+1} reduces to. Since $\mathcal{C}_{i+1}$ is finite, there is a pair $(G'^{\ell}_{i+1}, H'^{\ell}_{i+1})$ in $\mathrm{Red}(R_{i+1})$, such that neither G'_{i+1} nor H'_{i+1} is in $\mathcal{C}_{i+1}$. As before, since isomorphism of forests is polynomial-time solvable we can require that neither G'_{i+1} nor H'_{i+1} is a forest. We set $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \mathrm{Sup}(G'_{i+1}) \cup \mathrm{Sup}(H'_{i+1})$.

We define $\mathcal{C} = \bigcup_{i\in\mathbb{N}} \mathcal{C}_i$. We claim that $\mathcal{C}$ is an intermediate class. To argue this we define $\mathcal{D} = \bigcup_{i\in\mathbb{N}} \mathcal{D}_i$. By construction $\mathcal{C}_i$ and $\mathcal{D}_i$ are disjoint. Likewise $\mathcal{D}_i$ and $\mathcal{C}_{i+1}$ are disjoint. Therefore $\mathcal{C}$ and $\mathcal{D}$ are disjoint. If isomorphism for graphs in $\mathcal{C}$ were polynomial-time solvable then there would be an algorithm A_j solving all graphs in $\mathcal{C}$. However, $\mathcal{C}$ contains $\mathcal{C}_j$ which contains G_j and H_j and isomorphism of the pair (G^{ℓ}_j, H^{ℓ}_j) is not correctly decided by algorithm A_j (for some labeling of the graphs). It remains to argue that the class $\mathcal{C}$ is not isomorphism complete. If this were the case, there would be a reduction R_j such that all unlabeled versions of graphs in $\mathrm{Red}(R_j)$ are contained in $\mathcal{C}$. However, by construction the graphs G'_j and H'_j are in $\mathcal{D}_j$. In particular they are not contained in $\mathcal{C}$ but $(G'^{\ell}_j, H'^{\ell}_j)$ is in $\mathrm{Red}(R_j)$. This shows that $\mathcal{C}$ is intermediate. Since all $\mathcal{C}_i$ are closed under taking subgraphs, $\mathcal{C}$ is closed under taking subgraphs. □

For the construction in the proof it is essential that the girth among all supergraphs of a non-forest is bounded. Another way of stating this is that we can find an infinite anti-chain of graphs with respect to the subgraph relation by exploiting the girth. This explains why our construction will not work to construct an intermediate minor closed graph class. Furthermore, the proof requires that the class that does not contain any graph in $\mathcal{D}_i$ is graph isomorphism complete. This prevents us for example from applying the construction to obtain an intermediate graph class closed under topological minors. Of course the conditional existence of such intermediate graph classes closed under topological minors would imply that the assumption that graph isomorphism is not polynomial-time solvable is false.

References

1. Babai, L.: Automorphism groups, isomorphism, reconstruction. In: Handbook of Combinatorics, vol. 2, pp. 1447–1540. MIT Press, Cambridge (1995)
2. Booth, K.S., Colbourn, C.J.: Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Comput. Sci. Dep., Univ. Waterloo (1979)
3. Curtis, A.R., Lin, M.C., McConnell, R.M., Nussbaum, Y., Soulignac, F.J., Spinrad, J., Szwarcfiter, J.L.: Isomorphism of graph classes related to the circular-ones property. Discrete Mathematics & Theoretical Computer Science 15(1), 157–182 (2013)
4. Grohe, M.: Fixed-point definability and polynomial time on graphs with excluded minors. In: LICS, pp. 179–188 (2010)
5. Grohe, M., Marx, D.: Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In: STOC, pp. 173–192 (2012)
6. Köbler, J., Kuhnert, S., Verbitsky, O.: Helly circular-arc graph isomorphism is in logspace. In: Electronic Colloquium on Computational Complexity, ECCC (2013)
7. Köbler, J., Schöning, U., Torán, J.: The graph isomorphism problem: its structural complexity. Birkhäuser Verlag, Basel (1993)

8. Kratsch, S., Schweitzer, P.: Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. In: WG, pp. 34–45 (2012)
9. Ladner, R.E.: On the structure of polynomial time reducibility. J. ACM 22(1), 155–171 (1975)
10. Lueker, G.S., Booth, K.S.: A linear time algorithm for deciding interval graph isomorphism. J. ACM 26(2), 183–195 (1979)
11. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
12. Ponomarenko, I.N.: The isomorphism problem for classes of graphs that are invariant with respect to contraction. Zap. Nauchn. Sem. LOMI 174, 147–177 (1988) Russian. English translation in J. Soviet Math. 55, 1621–1643 (1991)
13. Schweitzer, P.: Problems of unknown complexity: Graph isomorphism and Ramsey theoretic numbers. PhD thesis, Universität des Saarlandes, Germany (2009)
14. Uehara, R.: Tractabilities and intractabilities on geometric intersection graphs. Algorithms 6(1), 60–83 (2013)

Determinantal Complexities and Field Extensions

Youming Qiao[1]*, Xiaoming Sun[2]**, and Nengkun Yu[3]

[1] Centre for Quantum Technologies, the National University of Singapore
jimmyqiao86@gmail.com

[2] Institute of Computing Technology, Chinese Academy of Sciences
sunxiaoming@ict.ac.cn

[3] State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.
Centre for Quantum Computation and Intelligent Systems (QCIS), Faculty of Engineering and Information Technology, University of Technology, Sydney, NSW 2007, Australia
nengkunyu@gmail.com

Abstract. Let $\mathbb{F}$ be a field of characteristic $\neq 2$. The *determinantal complexity* of a polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ is defined as the smallest size of a matrix M whose entries are linear polynomials of x_i's over $\mathbb{F}$, such that $P = \det(M)$ as polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. To determine the determinantal complexity of the permanent polynomial is a long-standing open problem.

Let $\mathbb{K}$ be an *extension field* of $\mathbb{F}$; then P can be viewed as a polynomial over $\mathbb{K}$. We are interested in the comparison between the determinantal complexity of P over $\mathbb{K}$ (denoted as $\mathtt{dc}_{\mathbb{K}}(P)$), and that of P over $\mathbb{F}$ (denoted as $\mathtt{dc}_{\mathbb{F}}(P)$). It is clear that $\mathtt{dc}_{\mathbb{K}}(P) \leq \mathtt{dc}_{\mathbb{F}}(P)$, and the question is whether strict inequality can happen. In this note we consider polynomials defined over $\mathbb{Q}$. For $P = x_1^2 + \cdots + x_n^2$, there exists a constant multiplicative gap between $\mathtt{dc}_{\mathbb{R}}(P)$ and $\mathtt{dc}_{\mathbb{C}}(P)$: we prove $\mathtt{dc}_{\mathbb{R}}(P) \geq n$ while $\lceil n/2 \rceil + 1 \geq \mathtt{dc}_{\mathbb{C}}(P)$. We also consider additive constant gaps: (1) there exists a quadratic polynomial $Q \in \mathbb{Q}[x, y]$, such that $\mathtt{dc}_{\mathbb{Q}}(Q) = 3$ and $\mathtt{dc}_{\overline{\mathbb{Q}}}(Q) = 2$; (2) there exists a cubic polynomial $C \in \mathbb{Q}[x, y]$ with a rational zero, such that $\mathtt{dc}_{\mathbb{Q}}(C) = 4$ and $\mathtt{dc}_{\overline{\mathbb{Q}}}(C) = 3$. For additive constant gaps, geometric criteria are presented to decide when $\mathtt{dc}_{\mathbb{Q}} = \mathtt{dc}_{\overline{\mathbb{Q}}}$.

1 Introduction

Let $\mathbb{F}$ be a field of characteristic $\neq 2$. In algebraic complexity theory, a polynomial $P(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ is *expressible as a determinant* of size s if there exists a matrix $M = (L_{i,j})_{i,j \in [s]}$ where $L_{i,j}$'s are affine forms (linear polynomials) in

* Research at the Centre for Quantum Technologies is funded by the Singapore Ministry of Education and the National Research Foundation.

** Part of this work was done while the author was visiting the Centre for Quantum Techologies, National University of Singapore. This work was supported in part by the National Natural Science Foundation of China Grant 61170062, 61222202.

$\mathbb{F}[x_1, \ldots, x_n]$, such that $P(x_1, \ldots, x_n) \equiv \det(M)$ as polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. A family of polynomials $P = \{P_n(x_1, \ldots, x_{t(n)}) \mid n \in \mathbb{N}, P_n \in \mathbb{F}[x_1, \ldots, x_{t(n)}]\}$, where $t(n)$ is a polynomial in n, is *explicit*[1] if: (1) $\deg(P_n)$ is polynomial in n; (2) the coefficient of a monomial in P_n can be computed in time polynomial in n. The question of interest is to study, for an explicit polynomial family $\{P_n\}_{n\in\mathbb{N}}$, the smallest $s(n)$ such that P_n can be expressed as determinant of size $s(n)$. $s(n)$ is called the *determinantal complexity* of P_n. The well-known permanent versus determinant conjecture proposed by Valiant [14], is that for permanent polynomials, asymptotically $s(n)$ is $\exp(\omega(\log^2 n))$. (Recall that the characteristic of $\mathbb{F}$ is not 2.) The best lower bound up-to-date is $\Omega(n^2)$ by [5, 11]. For an almost up-to-date survey on this problem, cf. Agrawal's survey [1].

Let $\mathbb{K}$ be an extension field of $\mathbb{F}$. Then a polynomial P over $\mathbb{F}$ can also be viewed as a polynomial over $\mathbb{K}$. It is natural to ask whether the determinantal complexity of P over $\mathbb{F}$ and that of P over $\mathbb{K}$ are the same. Let $\mathtt{dc}_{\mathbb{K}}(P)$ be the determinantal complexity of P over $\mathbb{K}$; similarly we have $\mathtt{dc}_{\mathbb{F}}(P)$. Clearly, $\mathtt{dc}_{\mathbb{K}}(P) \leq \mathtt{dc}_{\mathbb{F}}(P)$. The question is whether strict inequality can happen.

Broadly, how the underlying field affects the complexity in an algebraic computing model is a classical question that dates back to Strassen [12]; Bürgisser [2, Section 4.1] and Hrubeš and Yehudayoff [9] also contribute to the understanding of this problem. We survey their results in Section 1.2. However, none of the works mentioned provide concrete examples showing field extensions do reduce complexity. A lack of concrete examples may be explained as follows: to exhibit a gap between $\mathtt{dc}_{\mathbb{K}}$ and $\mathtt{dc}_{\mathbb{F}}$ where $\mathbb{K}$ is an extension field of $\mathbb{F}$, one needs a lower bound on $\mathtt{dc}_{\mathbb{F}}$. Then the lower bound technique for permanent in [4, 15], and the one in [5, 11] apply to fields (of a specific characteristic $\neq 2$) in a uniform way, presumably not appropriate for such a separation.

1.1 Our Results

In this note we make a first and modest step to show the separation of determinantal complexities due to field extensions. We consider polynomials defined over $\mathbb{Q}$ (rational polynomials), and the goal is to understand how much the determinantal complexity can be reduced by allowing field extensions (from $\mathbb{R}$ to $\mathbb{C}$, or from $\mathbb{Q}$ to the algebraic number field $\overline{\mathbb{Q}}$). We will use rational (algebraic, real, complex...) determinantal expressions whose meaning is self-explaining.

Our first result is a constant (≈ 2) multiplicative gap between $\mathtt{dc}_{\mathbb{C}}(P)$ and $\mathtt{dc}_{\mathbb{R}}(P)$ for $P = x_1^2 + \cdots + x_n^2 \in \mathbb{Q}[x_1, \ldots, x_n]$: it is easy to see $\mathtt{dc}_{\mathbb{C}}(P) \leq \lceil n/2 \rceil + 1$. On the other hand we prove

Theorem 1. $\mathtt{dc}_{\mathbb{R}}(x_1^2 + \cdots + x_n^2) \geq n$.

We also consider additive constant gaps. The reason to still record these results (in light of the constant multiplicative gap) is because in these cases, we are able to obtain exact geometric conditions on whether the polynomial has a (relatively) small expression over $\mathbb{Q}$ (cf. Proposition 2 and Theorem 7 for the description

[1] See [8] for details on this concept, in particular its relation with Valiant's VNP class.

of these conditions). The latter criterion (Theorem 7) is actually due to L. E. Dickson from 1920's. In the following Theorem 2 (2), over $\mathbb{Q}$, the $\mathtt{dc}_{\mathbb{Q}}(C) > 3$ lower bound is due to Dickson [6, Sec. 4] and is rather involved. We complement his result by providing an upper bound 4.

Theorem 2. *1. There exists quadratic $Q(x, y) \in \mathbb{Q}[x, y]$ s.t. $\mathtt{dc}_{\mathbb{Q}}(Q) = 3$ and $\mathtt{dc}_{\overline{\mathbb{Q}}}(Q) = 2$.*
2. There exists cubic $C(x, y) \in \mathbb{Q}[x, y]$ with a rational zero, s.t. $\mathtt{dc}_{\mathbb{Q}}(C) = 4$ and $\mathtt{dc}_{\overline{\mathbb{Q}}}(C) = 3$.

As mentioned, the interesting aspect of Theorem 2 is the connection with geometry: the criteria of having small rational expressions are closely related to algebraic geometry and algebraic curves. In Theorem 2 (1), for a bivariate quadratic polynomial Q, it turns out that modulo a certain degenerate case, Q has a rational expression of size 2 if and only if it has a zero with rational coordinates (a *rational zero*). In Theorem 2 (2), for a bivariate cubic polynomial C with a rational zero, Dickson presents a beautiful geometric criterion for whether $\mathtt{dc}_{\mathbb{Q}}(C) > 3$ (cf. Theorem 7).

1.2 Previous Works and Some Discussion

The works addressing this issue in algebraic complexity are by Strassen ([12], cf. [2, Sec. 4.1] and [3, Sec. 4.3]), Bürgisser [2] and Hrubeš and Yehudayoff [9]. Coincidentally, the authors sometimes come up with identical results independently (cf. e.g. [2, Prop 4.1 (iii)] and a part of [9, Theorem 4.2]), while the focuses can be different. In [3] the main results are concerned with when the extension fields do not reduce the complexity, under the name "autarky." In [2] the focus is on how field extensions affect Valiant's hypothesis. In [9], besides many other results, the authors also considered noncommutative extensions. The focus of the present work, is to provide concrete examples showing that the gaps indeed happen in the determinantal expression model.

In particular letting $L_{\mathbb{F}}(P)$ be the complexity of a polynomial $P \in \mathbb{Q}[x_1, \ldots, x_n]$ over an extension field $\mathbb{F}$ in general arithmetic circuits, Bürgisser asked whether $L_{\mathbb{Q}}(\mathsf{Perm}) = n^{\omega(1)}$ implies $L_{\overline{\mathbb{Q}}}(\mathsf{Perm}) = n^{\omega(1)}$, and posed this as an open problem [2, Problem 4.1]. It is also natural to pose this problem in the context of determinantal expressions.

Problem 1. For $P(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n]$, whether $\mathtt{dc}_{\mathbb{Q}}(P) = n^{\omega(1)}$ implies $\mathtt{dc}_{\overline{\mathbb{Q}}}(P) = n^{\omega(1)}$.

Two results in these works are of particular interest, namely (1) extensions of an algebraically closed field do not help (via Nullstellensatz, an observation by Strassen, cf. [2, Prop. 4.1 (i)]), and (2) for circuit complexity, going from an algebraic extension of degree d to the base field increases the complexity by a multiplicative factor $O(d^3)$ (cf. [2, Prop 4.1 (iii)] and [9, Theorem 4.2]). The first result applies to determinantal complexity. The latter result can also be adapted to determinantal expressions (with a slightly larger overhead) as follows: suppose $\mathbb{K}$ is an algebraic extension of $\mathbb{F}$ of degree d. First we convert the optimal determinantal expression over $\mathbb{K}$ of size s, to a skew circuit [10, 13]. Then apply the construction as in [2, Prop 4.1 (iii)] and [9, Theorem 4.2] to get another skew circuit over $\mathbb{F}$; the construction there preserves the skew circuit property with a little more care. Finally we convert back to determinantal expression, which is of size $O(d^3 s^4)$ by the conversions between skew circuits and determinantal expressions. This suggests the following natural strategy to tackle Problem 1 (and Bürgisser's problem). Given an optimal determinantal expression $\det(A_1 x_1 + \cdots + A_n x_n + A_0)$ of size s over $\overline{\mathbb{Q}}$ computing a polynomial $P(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n]$, consider the extension field $\mathbb{K}$ of $\mathbb{Q}$ by adjoining the algebraic numbers in A_i, $i \in \{0, 1, \ldots, n\}$. If the extension degree d of $\mathbb{K}$ over $\mathbb{Q}$ was bounded by $\mathsf{poly}(n, s)$, then $\mathsf{dc}_{\mathbb{Q}}(P) = O(d^3 s^4) = \mathsf{poly}(n, \mathsf{dc}_{\overline{\mathbb{Q}}}(P))$, showing a polynomial relation between $\mathsf{dc}_{\mathbb{Q}}$ and $\mathsf{dc}_{\overline{\mathbb{Q}}}$. However, to the best of our knowledge such a bound on the extension degree is not known.

Dickson contributed to some important statements in this note, in [6, 7] from more than 90 years ago. In fact, the results in [6, 7] are beyond the ones cited here. Motivated by geometric considerations and Diophantine analysis, he was able to determine all the (n, d) pairs, such that a *general* degree-d homogeneous polynomial in n variable can be expressed as determinantal expressions of size d with linear forms. For cubic homogeneous polynomials in 3 and 4 variables with at least one rational zeros, he could give out criteria to determine whether one such polynomial is expressible rationally of size 3. The 3-variate case is just the $\mathsf{dc}_{\mathbb{Q}}(C) > 3$ part in Theorem 2, while the 4-variate case involves spectacular calculation that spans over 18 pages.

In [11, Sec. 2], Mignon and Ressayre considered determinantal complexity of a homogeneous polynomial P *over algebraically closed fields*, when (1) the number of variables is ≤ 3; (2) $\deg(P) = 2$. These are just the cases treated here, but our focus is on the comparison with over non-algebraically closed fields.

Organization. We prove Theorem 1 in Section 3, showing a constant multiplicative gap. Section 4, 5 and 6 are devoted to prove Theorem 2, focusing on providing geometric/algabraic criteria for having small rational expressions.

2 Preliminaries

For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \ldots, n\}$. For a field $\mathbb{F}$, $\overline{\mathbb{F}}$ denotes the *algebraic closure* of $\mathbb{F}$. Let $\mathbb{K}$ be another field; $\mathbb{K}/\mathbb{F}$ means that $\mathbb{K}$ is an extension field of $\mathbb{F}$. For $S \subseteq \mathbb{K}$, $\mathbb{F}(S)$ denotes the smallest extension field of $\mathbb{F}$ that contains S, called the field *generated by S over* $\mathbb{F}$. Recall that $\alpha \in \mathbb{K}$ is algebraic over $\mathbb{F}$ if there exists $P(x) \in \mathbb{F}[x]$ such that $P(\alpha) = 0$; α is transcendental otherwise. A field extension $\mathbb{K}/\mathbb{F}$ is *algebraic* if every $\alpha \in \mathbb{K}$ is algebraic over $\mathbb{F}$; it is *transcendental* otherwise. Let $P(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$; P is *homogenous* if all monomials in it are of the same degree d. If P is not homogenous, then it can be *homogenized* by adding a fresh variable x_0 not in $\mathbf{x}$, and padding each monomial to degree d. The resulting homogeneous polynomial in $\mathbb{F}[x_0, \mathbf{x}]$ is denoted as $\overline{P}$. For $\mathbb{K}/\mathbb{Q}$, an *affine transformation* T over $\mathbb{K}$ consists of $M \in \mathrm{GL}(n, \mathbb{K})$ and $\mathbf{v} \in \mathbb{K}^n$; for $\mathbf{a} \in \mathbb{Q}^n$, $T(\mathbf{a}) = M\mathbf{a} + \mathbf{v}$. T is rational if it is over $\mathbb{Q}$. A *projective transformation* over $\mathbb{K}$ is $M \in \mathrm{GL}(n, \mathbb{K})$.

Observation 3. it For $P(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ of degree d, $P(\mathbf{x})$ is expressible of size d if and only if $\overline{P}(x_0, \mathbf{x})$ is expressible of size d.

Observation 4. itLet T be a rational affine transformation. $P(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ has a rational expression of size s if and only if $P(T(\mathbf{x}))$ has a rational expression of size s.

Fact 5. $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = \det \begin{pmatrix} 1 & & & & a_n x \\ -x & 1 & & & a_{n-1}x \\ & \cdots\cdots & & & \vdots \\ & & \cdots & 1 & a_2 x \\ & & & -x & a_1 x + a_0 \end{pmatrix}$,

where empty entries are 0.

3 The Constant Multiplicative Gap

For $P = x_1^2 + \cdots + x_n^2 \in \mathbb{Q}[x_1, \ldots, x_n]$, we exhibit an expression over $\mathbb{C}$ of size $n/2 + 1$ for even n. (This follows easily by the observation that P has a formula $(x_1 + ix_2)(x_1 - ix_2) + \cdots + (x_{n-1} + ix_n)(x_{n-1} - ix_n)$, where $i = \sqrt{-1}$, and utilize the reduction from formulas to determinantal expression as in [2, Proposition 2.30].) The odd n case follows by considering $n + 1$.

$$P(x_1, x_2, \ldots, x_n) = \det \begin{pmatrix} 0 & x_1 + ix_2 & x_3 + ix_4 & \cdots & x_{n-1} + ix_n \\ -(x_1 - ix_2) & 1 & 0 & \cdots & 0 \\ -(x_3 - ix_4) & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -(x_{n-1} - ix_n) & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

We now show $\mathtt{dc}_{\mathbb{R}}(P) \geq n$. First a lemma is required; recall that a linear form is a linear polynomial with constant term vanishing.

Lemma 1. *For $j \in [n]$, let $f_j(x_1, x_2, \cdots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$ be a linear form, and $g_j(x_1, x_2, \cdots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$. If $\sum_{i=1}^n x_i^2 = \sum_{j=1}^m f_j g_j$, then $m \geq n$.*

Proof. We prove this lemma by induction. It is trivial when $n = 1$. Now suppose it is true for $n = k - 1$, and we know that

$$\sum_{i=1}^k x_i^2 = \sum_{j=1}^m f_j(x_1, x_2, \cdots, x_k) g_j(x_1, x_2, \cdots, x_k).$$

Without loss of generality, assume that $f_1(x_1, x_2, \cdots, x_n) = \sum_{i=1}^k a_i x_i$ is nonzero and $a_1 \neq 0$. Setting $x_1 = -(\sum_{i>1} a_i x_i)/a_1 := h(x_2, x_3, \cdots, x_k)$, $f_1(x_1, \cdots, x_k) = 0$, and

$$h^2(x_2, x_3, \cdots, x_k) + \sum_{i=2}^k x_i^2 = \sum_{j=2}^m f_j(h, x_2, \cdots, x_k) g_j(h, x_2, \cdots, x_k).$$

Note that $h^2(x_2, x_3, \cdots, x_k) + \sum_{i=2}^k x_i^2$ is a quadratic form in $x_2, \cdots, x_k$ with real coefficients, and it is positive definite. By positive definiteness, there exist $B = (b_{ij}) \in \mathbb{R}^{(k-1)\times(k-1)}$, $i, j \in \{2, \ldots, k\}$, B nonsingular, s.t. by setting $x_i = \ell_i(y_2, \ldots, y_k) := \sum_{j=2}^k b_{ij} y_j$ where y_j's are new variables, we have $h^2(x_2, x_3, \cdots, x_k) + \sum_{i=2}^k x_i^2 = \sum_{j=2}^k y_i^2$. This gives

$$\sum_{j=2}^{k} y_i^2 = \sum_{j>1}^{m} f_j(h(\ell_2, \ell_3, \cdots, \ell_k), \ell_2, \cdots, \ell_k) g_j(h(\ell_2, \ell_3, \cdots, \ell_k), \ell_2, \cdots, \ell_k).$$

Note that for $j > 1$, $f_j(h(\ell_2, \ell_3, \cdots, \ell_k), \ell_2, \cdots, \ell_k) = u_j(y_2, y_3 \cdots, y_k)$ is a linear form in $y_2, y_3 \cdots, y_k$ with real coefficients, and $g_j(h(\ell_2, \ell_3, \cdots, \ell_k), \ell_2, \cdots, \ell_k)$ is a polynomial of $(y_2, y_3 \cdots, y_k)$ with real coefficients. By the induction hypothesis, $m - 1 \geq k - 1$, that is $m \geq n$. This completes the proof.

Theorem 1, restated. $\mathtt{dc}_{\mathbb{R}}(x_1^2 + \cdots + x_n^2) \geq n$.

Proof. Suppose there exist real $m \times m$ matrices $A_1, \cdots, A_n$ and B such that $\sum_{i=1}^n x_i^2 = \det(\sum_{i=1}^n x_i A_i + B)$. First, we note that $\det(B) = 0$. Then there exist real non-singular matrices P and Q such $B = PDQ$ where D is a diagonal matrix with the elements of the first row being 0. Therefore

$$x_1^2 + \cdots + x_n^2 = \det(\sum_{i=1}^{n} x_i A_i + B) = \det(PQ) \det(\sum_{i=1}^{n} x_i P^{-1} A_i Q^{-1} + D).$$

The rest part of the proof is to expand $\det(\sum_{i=1}^n x_i P^{-1} A_i Q^{-1} + D)$ along the first row and invoke the previous lemma.

4 Bivariate Quadratic Polynomials

In this section we prove Theorem 2 (1), by showing the following: let Q be a quadratic polynomial in $\mathbb{Q}[x, y]$.

1. (Upper bound, Section 4.1) $Q(x, y)$ is expressible algebraically of size 2, and rationally of size 3.
2. (Lower bound, Section 4.2) There exists a quadratic $Q(x, y)$ s.t. $\mathtt{dc}_{\mathbb{Q}}(Q) > 2$.

4.1 Expressibility

We set up some notations first. The target is the polynomial

$$Q(x, y) = ax^2 + by^2 + cxy + dx + ey + f \in \mathbb{Q}[x, y]. \tag{1}$$

A useful representation of $Q(x, y)$ is

$$Q(x, y) = (ax + cy + d)x + (by^2 + ey + f). \tag{2}$$

Note that $Q(x, y)$ can be expressed as $\det \begin{pmatrix} L_1 & -R_1 \\ R_2 & L_2 \end{pmatrix}$ where L_i, R_i are affine forms if and only if $Q(x, y) = L_1 L_2 + R_1 R_2$. Thus by Equation 2, if $b = 0$ or $f = 0$, $by^2 + ey + f$ is a product of two affine forms in $\mathbb{Q}[x, y]$, which shows:

Observation 6. it If $a = 0$ or $b = 0$ or $f = 0$, then $Q(x, y)$ can be expressed rationally of size 2.

Proposition 1. *$Q(x, y)$ is expressible algebraically of size 2, and rationally of size 3.*

Proof. By Observation 6 we can assume that a, b, f are nonzero. Consider $by^2 + ey + f$ as in Equation 2, where $b \neq 0$. Let r_1, r_2 in $\overline{\mathbb{Q}}$ be the two roots of $y^2 + (e/b)y + (f/b)$, then Equation 2 can be written as $(ax + cy + d)x + (by - br_1)(y - r_2)$, which gives an algebraic expression of size 2.

We construct a size-3 rational expression for $Q(x, y)$. One construction is as follows.

$$ax^2 + by^2 + cxy + dx + ey + f = \det \begin{pmatrix} 1 & 0 & ax + \frac{c}{2}y \\ 0 & 1 & \frac{c}{2}x + by \\ -x & -y & dx + ey + f \end{pmatrix}. \tag{3}$$

Here's another construction for $c \neq 0$; let $t = \frac{d}{c} - \frac{ea}{c^2}$.

$$ax^2 + by^2 + cxy + dx + ey + f = \det \begin{pmatrix} 1 & 0 & y \\ by & cx + e & (-1)(f - et) \\ -1 & 1 & \frac{a}{c}x + t \end{pmatrix}. \tag{4}$$

Then, if $c = 0$, an appropriate affine transformation (like $x \to x + y$) can be performed to make it nonzero.

4.2 Inexpressibility

We present the existence of bivariate quadratic polynomials with $\mathtt{dc}_{\mathbb{Q}} > 2$. Two proofs will be given: the first one (Claim 4.2) is from the algebraic perspective, while the other (Proposition 2) gives a geometric criterion of whether a size-2 rational expression exists. The reason to present both proofs is that they represent two approaches for this problem respectively: on one hand, by algebraic manipulations, the problem would reduce to the rational solvability of another polynomial (usually with higher degree). On the other hand, clean geometric criterion may exist. For a more complicated example of the algebraic approach, cf. [6, pp. 114, Theorem]; Theorem 7, proved also by Dickson, is a more complicated example of the geometric criterion. For a concrete example, $x^2 + y^2 + 1$ is not expressible rationally of size 2.

Algebraic Approach. We first recall a classical representation of $Q(x, y) = ax^2 + by^2 + cxy + dx + ey + f \in \mathbb{Q}[x, y]$. Let $\mathbf{z} = (x, y, 1)^T$, then $Q(x, y)$ can be written as $\mathbf{z}^T A \mathbf{z}$ where A is $\begin{pmatrix} a & c/2 & d/2 \\ c/2 & b & e/2 \\ d/2 & e/2 & f \end{pmatrix}$. While in fact, any matrix $A(u, v, w)$ of the form $\begin{pmatrix} a & u & v \\ c - u & b & w \\ d - v & e - w & f \end{pmatrix}$, where u, v, w are formal variables that can serve to designate $Q(x, y)$.

Claim. $Q(x, y)$ has a size-2 rational expression if and only if $\det(A(u, v, w)) = 0 \in \mathbb{Q}[u, v, w]$ has a rational solution.

Proof. **($\Leftarrow$):** Suppose that $Q(x, y) = L_1L_2 + R_1R_2$, where $L_i(x, y) = \ell_{i1}x + \ell_{i2}y + \ell_{i3}$ and $R_i(x, y) = r_{i1}x + r_{i2}y + r_{i3}$, $L_i, R_i \in \mathbb{Q}[x, y]$. Denote $\mathbf{l}_i = (\ell_{i1}, \ell_{i2}, \ell_{i3})$ and $\mathbf{r}_i = (r_{i1}, r_{i2}, r_{i3})$. Write $L_1L_2 = (\mathbf{z}^T \cdot \mathbf{l}_1)(\mathbf{l}_2^T \cdot \mathbf{z}) = \mathbf{z}^T(\mathbf{l}_1\mathbf{l}_2^T)\mathbf{z}$, and similarly we have $R_1R_2 = \mathbf{z}^T(\mathbf{r}_1\mathbf{r}_2^T)\mathbf{z}$. It follows that $L_1L_2 + R_1R_2 = \mathbf{z}^T(\mathbf{l}_1\mathbf{l}_2^T + \mathbf{r}_1\mathbf{r}_2^T)\mathbf{z}$. From the matrix representation of $Q(x, y)$, it must be the case that $\mathbf{l}_1\mathbf{l}_2^T + \mathbf{r}_1\mathbf{r}_2^T = A(u, v, w)$ for certain $u, v, w \in \mathbb{Q}$. By the definition of the matrix rank, $A(u, v, w)$ is singular thus $\det(A(u, v, w)) = 0$ has a rational solution.
($\Rightarrow$): if $\det(A(u, v, w)) = 0$ has a rational solution (p, q, r), then $A(p, q, r)$ is of rank ≤ 2 over $\mathbb{Q}$ thus can be written as $\mathbf{l}_1\mathbf{l}_2^T + \mathbf{r}_1\mathbf{r}_2^T$ where $\mathbf{l}_i, \mathbf{r}_i \in \mathbb{Q}^3$.

Geometric Approach. Let us start with an observation. Suppose $Q(x, y)$ is expressed rationally of size 2 by $M = \begin{pmatrix} L_1 & -R_1 \\ R_2 & L_2 \end{pmatrix}$. Let $L_i = \ell_{i1}x + \ell_{i2}y + \ell_{i3}$, and $R_i = r_{i1}x + r_{i2}y + r_{i3}$. Consider the linear system $V_1 = Z(L_1 = 0, -R_1 = 0)$. If $\begin{pmatrix} \ell_{11} & \ell_{12} \\ r_{11} & r_{12} \end{pmatrix}$ is of full rank, that is, V_1 is nonsingular, then there is a rational zero in V_1, which is also a rational zero on $Q(x, y)$. This argument can be applied similarly to $V_2 = Z(L_1 = 0, R_2 = 0)$, $V_3 = Z(R_2 = 0, L_2 = 0)$ and $V_4 = Z(-R_1 = 0, L_2 = 0)$. If all of V_i's are singular, then $-R_1 = c_2L_1 + d_2$, $R_2 = c_3L_1 + d_3$ and $L_2 = c_4L_1 + d_4$, for $c_i, d_i \in \mathbb{Q}$. Thus $Q(x, y)$ can be written as $rL_1^2 + sL_1 + t$, for $r, s, t \in \mathbb{Q}$. We define $Q(x, y)$ to be *degenerate* if it can be written as such. That is, if P has a size-2 rational expression, then either it has a rational zero, or it is degenerate. Conversely, if $Q(x, y)$ is degenerate, then $Q(x, y)$ always has a size-2 rational expression. Also, if $Q(x, y)$ has a rational zero, then by a rational shift we can eliminate the constant term. Then a size-2 rational expression exists due to Observation 6. We now summarize the above as the following proposition.

Proposition 2. *$Q(x, y)$ can be expressed rationally as size 2, if and only if either $Q(x, y)$ is degenerate, or $Q(x, y)$ has a rational zero.*

In [7], Dickson showed that the homogenized version of P can be expressible rationally of size 2 if and only if it has a rational point in the projective space.

5 General Bivariate Cubic Polynomials

A bivariate cubic polynomial is:

$$D(x, y) = ax^2 + by^2 + cxy + dx + ey + f + gx^3 + hx^2y + ixy^2 + jy^3 \in \mathbb{Q}[x, y], \quad (5)$$

where at least one of g, h, i, j is nonzero.

Proposition 3. *$D(x, y)$ can be expressed rationally of size at most 5.*

Proof. We can assume $g \neq 0$ by performing appropriate rational transformations. Consider $D(x, y) = (gx^3 + ax^2 + dx + f) + y(jy^2 + hx^2 + ixy + cx + by + e)$. By Fact 5 (with some row and column permutations), we have $gx^3+ax^2+dx+f = \det\begin{pmatrix} -x & 0 & 1 \\ 1 & -x & 0 \\ ax & dx+f & gx \end{pmatrix}$. Changing Equation 3 slightly, we get $jy^2 + hx^2 + ixy + cx + by + e = \det\begin{pmatrix} 0 & -1 & y \\ 1 & 0 & -\frac{h}{g}x \\ gx & ix+jy & cx+by+e \end{pmatrix}$. Merging the above identities carefully, we get $D(x, y) = \det\begin{pmatrix} 0 & -y & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & y \\ -x & 0 & 1 & 0 & -\frac{h}{g}x \\ 1 & -x & 0 & 0 & 0 \\ ax & dx+f & gx & ix+jy & cx+by+e \end{pmatrix}$.

The following proposition is due to Dickson.

Proposition 4 (Dickson, [7]). *$D(x, y)$ has an algebraic expression of size* 3.

6 Bivariate Cubic Polynomials with a Rational Zero

In this section we prove Theorem 2 (2), by showing the following: let C be a cubic polynomial in $\mathbb{Q}[x, y]$ s.t. C has a rational zero.

1. (Upper bound, Section 6.1) $C(x, y)$ is expressible rationally of size 4;
2. (Lower bound, Dickson [6], Section 6.2) There exists $C(x, y)$ s.t. $\mathtt{dc}_{\mathbb{Q}}(C) > 3$.

Recall that by Proposition 4 $C(x, y)$ can be expressed algebraically of size 3.

6.1 Expressibility

Let D be given as in Equation 5. By a rational affine transformation, we can assume that $(0, 0)$ is a zero on $D(x, y)$. Thus f term can be removed from D and we get the target polynomial in this section:

$$C(x, y) = ax^2 + by^2 + cxy + dx + ey + gx^3 + hx^2y + ixy^2 + jy^3. \qquad (6)$$

By fixing $(0, 0)$ as a zero of $C(x, y)$ we can not perform rational transformations that violate so.

Claim. $C(x, y)$ has a rational expression of size 4.

Proof. Write $C(x, y)$ as $C(x, y) = x(gx^2+ax+d)+y(jy^2+hx^2+ixy+cx+by+e)$. We can assume $i, j \neq 0$ by performing a rational affine transformation like $x \to x + \lambda y$ (or $y \to x + \lambda y$), for an appropriate $\lambda \in \mathbb{Q}$. Let $t = \frac{c}{i} - \frac{dj}{i^2}$. By Equation 4, we have $jx^2 + hy^2 + ixy + cx + dy + e = \det\begin{pmatrix} 1 & 0 & y \\ hy & ix+d & (-1)(e-dt) \\ -1 & 1 & \frac{j}{i}x + t \end{pmatrix}$. We shall express $C(x, y)$ in the form of $\begin{pmatrix} y & \ell & m & n \\ 0 & 1 & 0 & y \\ x & hy & ix+d & (-1)(e-dt) \\ 0 & -1 & 1 & \frac{j}{i}x+t \end{pmatrix}$ where ℓ, m, n are affine forms to be determined. Its determinant evaluates to

$$y(jx^2 + hy^2 + ixy + cx + dy + e) - x \cdot [m(\frac{j}{i}x + t) - n + y(\ell + m)].$$

Thus setting $\ell = -m$, $m = -\frac{ig}{j}x$ and $n = -[(\frac{igt}{j} - a)x - d]$, we see that $m(\frac{j}{i}x + t) - n + y(\ell - m) = -(gx^2 + ax + d)$.

6.2 Dickson's Inexpressibility Criterion

In this section we describe the geometric criterion of whether or not C has a rational expression of size 3 by Dickson. He dealt with the homogeneous version of C; this is legitimate by Observation 3. Homogenizing Equation 6 we get

$$\overline{C}(x, y, z) = z^2(dx + ey) + z(ax^2 + by^2 + cxy) + (gx^3 + hx^2y + ixy^2 + jy^3).$$

Note that $(0 : 0 : 1)$ is a zero of $\overline{C}$.

We first show that if $\overline{C}$ is reducible over $\mathbb{Q}$ then it is expressibly rationally of size 3. Suppose $\overline{C} = \ell \cdot Q$ where ℓ and Q are linear and quadratic forms in x, y, z, respectively. By a projective transformation let $\ell \to y$; then $\overline{C} = y \cdot Q'$. If x^2 does not appear in Q, then by Observation 6 Q is expressible rationally of size 2, thus $\overline{C}$ is expressible rationally of size 3. Otherwise by a projective transformation on x we eliminate xy and xz terms to get $\overline{C} = y \cdot (x^2 + \alpha y^2 + \beta yz + \gamma z^2)$. Finally use $y(x^2 + \alpha y^2 + \beta yz + \gamma z^2) = \det \begin{pmatrix} x & y & 0 \\ -\alpha y - \beta z & x & \gamma z \\ z & 0 & y \end{pmatrix}$ to conclude.

From now on we shall assume that $\overline{C}$ is irreducible. Before stating the criterion for irreducible $\overline{C}$ we need to recall two concepts for algebraic curves.

Let $R \in \mathbb{Q}[x, y, z]$ be a homogeneous polynomial of d; it defines a projective plane curve C_R. The *Hessian* of R is $H_R(x,y,z) = \begin{pmatrix} \frac{\partial^2 P}{\partial x \partial x} & \frac{\partial^2 P}{\partial x \partial y} & \frac{\partial^2 P}{\partial x \partial z} \\ \frac{\partial^2 P}{\partial y \partial x} & \frac{\partial^2 P}{\partial y \partial y} & \frac{\partial^2 P}{\partial y \partial z} \\ \frac{\partial^2 P}{\partial z \partial x} & \frac{\partial^2 P}{\partial z \partial y} & \frac{\partial^2 P}{\partial z \partial z} \end{pmatrix}$. $\det(H_R)$ is a polynomial in $\mathbb{Q}[x, y, z]$ with degree $3(d - 2)$. Then a nonsingular point $(a : b : c) \in C_R$ is an *inflection point* if and only if $\det H_R(a, b, c) = 0$. [2] If R is of degree 3, let O be a nonsingular point on C_R but not an inflection point. By Bézout's theorem, the tangent at O to C_R must intersect C_R at a point distinct from O. This new point is called the *tangential* of O w.r.t. C_R.

Then we can state the theorem by Dickson.

Theorem 7 (Dickson, [6]). *Let $\overline{C}$ be an irreducible homogeneous cubic bivariate polynomial with a rational point O. Then $\overline{C}$ is expressible rationally of size 3 if and only if*

[2] Recall that geometrically, a nonsingular point O on a curve C is an inflection point if the intersection of C and the tangent line to O is of multiplicity ≥ 3.

- *either O is an inflection point, and $\overline{C}$ has a further rational point not on $y = 0$;*
- *or O is not an inflection point, and there exists a line $\overline{L}$, such that: (1) $\overline{L}$ is not the tangent to O; (2) $\overline{L}$ passes the tangential of O w.r.t. $\overline{C}$; (3) $\overline{L}$ intersects with $\overline{C}$ at 3 rational points.*

Acknowledgement. We would like to thank the anonymous reviewers for helpful comments on a preliminary version of this note. In particular, one reviewer suggested to consider the construction in Section 1.2.

References

1. Agrawal, M.: Determinant versus permanent. In: Proceedings of the International Congress of Mathematicians (ICM), pp. 1409–1421 (2006)
2. Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory. In: Algorithms and Computation in Mathematics. Springer (2000)
3. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. A series of comprehensive studies in mathematics. Springer (1997)
4. Cai, J.-Y.: A note on the determinant and permanent problem. Information and Computation 84(1), 119–127 (1990)
5. Cai, J.-Y., Chen, X., Li, D.: A quadratic lower bound for the permanent and determinant problem over any characteristic $\neq 2$. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 491–498 (2008)
6. Dickson, L.E.: Algebraic theory of the expressibility of cubic forms as determinants, with application to diophantine analysis. American Journal of Mathematics 43(2), 102–125 (1921)
7. Dickson, L.E.: Determination of all general homogeneous polynomials expressible as determinants with linear elements. Transactions of the American Mathematical Society 22(2), 167–179 (1921)
8. Hrubes, P., Wigderson, A., Yehudayoff, A.: Relationless completeness and separations. In: IEEE Conference on Computational Complexity, pp. 280–290 (2010)
9. Hrubes, P., Yehudayoff, A.: Arithmetic complexity in ring extensions. Theory of Computing 7(1), 119–129 (2011)
10. Malod, G., Portier, N.: Characterizing valiant's algebraic complexity classes. J. Complex. 24(1), 16–38 (2008)
11. Mignon, T., Ressayre, N.: A quadratic bound for the determinant and permanent problem. In: International Mathematics Research Notices, pp. 4241–4253 (2004)
12. Strassen, V.: Berechnung und programm. I. Acta Inf. 1, 320–335 (1972)
13. Toda, S.: Classes of arithmetic circuits capturing the complexity of computing the determinant. IEICE Trans. Inf. Syst. E75-D, 116–124 (1992)
14. Valiant, L.G.: Completeness classes in algebra. In: Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, STOC 1979, pp. 249–261. ACM, New York (1979)
15. von, J., von zur Gathen, J.: Permanent and determinant. Linear Algebra and its Applications 96, 87–100 (1987)

Algorithms to Measure Diversity and Clustering in Social Networks through Dot Product Graphs

Matthew Johnson[1], Daniël Paulusma[1], and Erik Jan van Leeuwen[2]

[1] School of Engineering and Computer Science, Durham University, England
{matthew.johnson2,daniel.paulusma}@durham.ac.uk
[2] Max-Planck Institut für Informatik, Saarbrücken, Germany
erikjan@mpi-inf.mpg.de

Abstract. Social networks are often analyzed through a graph model of the network. The *dot product model* assumes that two individuals are connected in the social network if their attributes or opinions are similar. In the model, a d-dimensional vector $\mathbf{a}^v$ represents the extent to which individual v has each of a set of d attributes or opinions. Then two individuals u and v are assumed to be friends, that is, they are connected in the graph model, if and only if $\mathbf{a}^u \cdot \mathbf{a}^v \geq t$, for some fixed, positive threshold t. The resulting graph is called a *d-dot product graph.*

We consider two measures for diversity and clustering in social networks by using a d-dot product graph model for the network. Diversity is measured through the size of the largest independent set of the graph, and clustering is measured through the size of the largest clique. We obtain a tight result for the diversity problem, namely that it is polynomial-time solvable for $d = 2$, but NP-complete for $d \geq 3$. We show that the clustering problem is polynomial-time solvable for $d = 2$. To our knowledge, these results are also the first on the computational complexity of combinatorial optimization problems on dot product graphs.

We also consider the situation when two individuals are connected if their preferences are not opposite. This leads to a variant of the standard dot product graph model by taking the threshold t to be zero. We prove in this case that the diversity problem is polynomial-time solvable for any fixed d.

1 Introduction

Social networks are often modeled by a graph in order to use advanced algorithmic (or statistical) tools. Indeed, there is a large body of literature on (random) graph models for social networks (see, for example, the surveys by Newman [23] and Snijders [32]). Many of these studies verify that a particular model has properties that have been observed in real-world social networks, such as a power-law degree distribution or the small-world principle, but do not consider why connections are made in the first place. This has led to the development of models that do take such reasons into account (a partial overview is in Liben-Nowell and Kleinberg [21]). For example, the models of Simon [31], Price [26], and Barabási and Albert [3] famously pose that if you have many friends, you are more likely to befriend more people. A similar type of engagement was recently considered from an algorithmic perspective by Bhawalkar et al. [6].

We consider a different predictor for connections in a social network, namely the degree of similarity of attributes and opinions of different individuals. Generally, individuals with similar attributes or opinions are more likely to be connected. This is known as the *homophily principle* and has a long tradition within sociological research (see, for example, the survey by McPherson et al. [22]). To model the attributes of an individual u, we can associate them with a vector $\mathbf{a}^u$, where an entry a_i^u expresses the extent to which u has an attribute or opinion i [33]. For example, a positive value of a_i^u could indicate that u likes item i, whereas a negative value suggests that u dislikes item i. We call this a *vector model.*

There are many ways to measure similarity using a vector model (see, for example, [1,14,19,33]). We will use the dot product as a similarity measure, leading to the *dot product model* for social networks. Formally, this model is defined as follows. Consider a social network that consists of a set V of individuals, together with a vector model $\{\mathbf{a}^u \mid u \in V\}$. Let

$$\mathrm{sim}(u,v) = \mathbf{a}^u \cdot \mathbf{a}^v = \textstyle\sum_{i=1}^{d} a_i^u a_i^v.$$

If the similarity $\mathrm{sim}(u,v)$ is at least some specified *threshold* $t > 0$, then we view the preferences of u and v to be sufficiently close together for u and v to be connected, that is, to be friends within the network. This immediately implies a graph $G = (V,E)$, where $(u,v) \in E$ if and only if $\mathrm{sim}(u,v) \geq t$. Such a graph is called a *dot product graph* of *dimension* d, or a d*-dot product graph.* The vector model $\{\mathbf{a}^u \mid u \in V\}$ together with the threshold t is called a d*-dot product representation* of G.

The dot product graph as a model for social networks was recently formalized by Nickel, Scheinerman, Tucker, and Young [24,30,34,35]. Their studies were motivated by earlier work of Papadimitriou et al. [25] and Caldarelli et al. [7]. However, dot product graphs have a much longer tradition, both in sociology (see, for example, Breiger [5]) and in graph theory. We briefly survey known graph-theoretic results. Reiterman et al. [27,28,29] and particularly Fiduccia et al. [10] proved several structural results. The work of Fiduccia et al. [10] implies that 1-dot product graphs can be recognized in polynomial time. However, Kang and Müller [17] showed the problem of deciding whether a graph has dot product dimension d is NP-hard for all fixed $d \geq 2$ (membership of NP is still open). They also proved that an exponential number of bits is sufficient and can be necessary to store a d-dot product representation of a dot product graph. There are several papers that consider the minimum dimension d such that a graph is a d-dot product graph (the *dot product dimension* of a graph) [16,20], deriving for example a tight bound of 4 on the dot product dimension of a planar graph [16]. Finally, dot product graphs share some ideas with low-complexity graphs [2].

In this paper, we consider the complexity of computing advanced structural measures of social networks through the dot product model. Note that many standard structural measures, such as the graph diameter and the clustering coefficient, are easy to compute. Therefore, we consider two more advanced measures for diversity and clustering. These are related to classic graph optimization problems whose computational complexity on dot product graphs was unknown. In fact, to the best of our knowledge, our work provides the first complexity results for graph optimization problems on dot product graphs.

First, we consider a measure for diversity, by finding (the size of) a largest group of individuals in the network that are different-minded, and thus pairwise disconnected. This corresponds to the well-known Independent Set problem, which is NP-complete, W[1]-complete, and very hard to approximate on general graphs [18,9,13], but its complexity on dot product graphs is open. We settle this by proving that Independent Set is polynomial-time solvable on 2-dot product graphs, but becomes NP-complete on 3-dot product graphs.

Second, we consider a measure for clustering, by finding (the size of) a largest group of individuals in the network that are like-minded, and thus pairwise connected. This corresponds to the well-known Clique problem, which is also NP-complete, W[1]-complete, and very hard to approximate on general graphs [18,9,13], but its complexity has not been analyzed on dot product graphs. We give initial insights into the complexity of this problem and show that it is polynomial-time solvable on 2-dot product graphs.

To complement these results, we consider two variants of the dot product model. For the first variant, we model the scenario in which two individuals are connected if their preferences are not opposite. That is, consider the graph where two individuals u, v are connected if and only if $\mathbf{a}^u \cdot \mathbf{a}^v \geq 0$. We call such a graph a *d^0-dot product graph*. Recall that in d-dot product graphs, the threshold t for connectivity must be greater than zero, and hence the definition of d^0-dot product graphs is different. Moreover, the structure of d^0-dot product graphs is substantially different from that of d-dot product graphs. To illustrate this, we prove that Independent Set is polynomial-time solvable on d^0-dot product graphs for any fixed d and that Clique is polynomial-time solvable if $d \leq 3$.

For the second variant, we model the situation in which two individuals are connected in the model if their preferences are neither opposite nor orthogonal. Consider the graph that is obtained when two vertices u, v are adjacent if and only if $\mathbf{a}^u \cdot \mathbf{a}^v > 0$. We call this a *$d^+$-dot product graph*. It follows from Fiduccia et al. [10] that the graph class where two vertices are adjacent if and only if $\mathbf{a}^u \cdot \mathbf{a}^v > t$ for some $t > 0$ is equivalent to the class of d-dot product graphs. However, we prove that the structure of d^+-dot product graphs is different from that of d-dot product graphs and that of d^0-dot product graphs. Still, we can show that Independent Set is polynomial-time solvable on d^0-dot product graphs for any fixed d, as is Clique when $d \leq 3$.

We provide an overview of our results in Table 1.

2 Preliminaries

All graphs that we consider are finite, undirected, and have neither loops nor multiple edges. For undefined graph terminology we refer to Diestel [8].

Table 1. An overview of our results for the problems Independent Set and Clique on d-dot product graphs (the first row), d^0-dot product graphs (the second row), and d^+-dot product graphs (the third row), respectively, for fixed dimension d.

Setting	Independent Set	Clique
d-DPG (≥ 1)	in P for $d \leq 2$ NP-complete for $d \geq 3$	in P for $d \leq 2$? for $d \geq 3$
d^0-DPG (≥ 0)	in P for $d \geq 0$	in P for $d \leq 3$? for $d \geq 4$
d^+-DPG (> 0)	in P for $d \geq 0$	in P for $d \leq 3$? for $d \geq 4$

Let $G = (V, E)$ be a graph. We denote the neighbourhood of a vertex $u \in V$ by $N(u) = \{v \mid (u, v) \in E\}$. A subset $U \subseteq V$ is *independent* if no two vertices in U are joined by an edge, and U is a *clique* if every two vertices of U are adjacent. Given $U \subseteq V$, $G[U]$ denotes the subgraph of G induced by U, that is, it has vertex set U and an edge between two vertices of U if and only if G has an edge between them. The *complement* of G has vertex set V and an edge between two distinct vertices if and only if these vertices are not adjacent in G.

A graph is a *comparability graph* if there exists an assignment of exactly one direction to each of its edges such that (a, c) is a directed edge whenever (a, b) and (b, c) are directed edges. The complement of a comparability graph is called a *co-comparability graph.* A graph is *p-partite* if its vertex set can be partitioned into at most p independent sets. If $p = 2$, then the graph is called *bipartite.* The complement of a p-partite graph is called a co-p-partite graph. Observe that the vertex set of a co-p-partite graph can be partitioned into at most p cliques. The complement of a bipartite graph is called *co-bipartite.*

3 Structure of *d*-Dot Product Graphs

In this section, we describe some of the structure of d-dot product graphs, which we need in our algorithms later on. Fiduccia et al. [10, Theorem 20] proved that 1-dot product graphs have at most two nontrivial components, each of which are threshold graphs. We show that d-dot product graphs, and in particular 2-dot product graphs, exhibit similar interesting structural properties.

From now we assume that $d \geq 2$. The reason for doing this is that our polynomial-time results on Independent Set and Clique in Section 4 for the case $d = 2$ readily carry over to the case $d = 1$: we can represent a $(d-1)$-dot product graph as a d-dot product graph for all $d \geq 2$ by adding a zero entry to all vectors of any of its $(d-1)$-dot product representations.

We call a d-dot product representation of a graph *clean* if it contains no two vectors $\mathbf{a}^u$ and $\mathbf{a}^v$ with $\mathbf{a}^u = \gamma \mathbf{a}^v$ for some $\gamma \geq 0$.

Lemma 1. ($\bigstar$)[1] *Given a d-dot product graph G without isolated vertices and a d-dot product representation of G, we can compute a clean d-dot product representation of G in polynomial time.*

[1] Proofs marked with a star have been omitted due to page restrictions.

Throughout the remainder of this section, we assume that we are given a d-dot product graph $G = (V, E)$ for some $d \geq 2$ together with a d-dot product representation with vectors $\{\mathbf{a}^u \mid u \in V\}$ and threshold t. For solving Independent Set and Clique, we can preprocess G by removing any isolated vertices. Hence, by Lemma 1, we may assume without loss of generality that the given representation is clean.

We will use the notation θ_{uv} for the angle between $\mathbf{a}^u$ and $\mathbf{a}^v$, which is the smaller of the two angles between $\mathbf{a}^u$ and $\mathbf{a}^v$ in the plane defined by $\mathbf{a}^u$ and $\mathbf{a}^v$. We assume some fixed direction of rotation so $\theta_{uv} = -\theta_{vu}$.

We say that a vertex u is *short* if $||\mathbf{a}^u|| \leq \sqrt{t}$; otherwise, it is *long.* Note that we can decide whether u is short in polynomial time by checking whether $||\mathbf{a}^u||^2 \leq t$. We first provide two lemmas about short vertices.

Lemma 2. (★) *Let v be a short vertex. Then $G[N(v)]$ is co-2^{d-1}-partite.*

The lemma shows in particular that $G[N(v)]$ is co-bipartite if $d = 2$.

Lemma 3. (★) *The set of short vertices is an independent set.*

We say that a vertex v is *between* vertices u and w if $\boldsymbol{a}^v$ can be written as a nonnegative linear combination of $\boldsymbol{a}^u$ and $\boldsymbol{a}^w$. In other words, v is between u and w if $\mathbf{a}^v$ lies in the plane defined by $\mathbf{a}^u$ and $\mathbf{a}^w$ and $\boldsymbol{a}^v$ lies within the smaller of the two angles defined by $\boldsymbol{a}^u$ and $\boldsymbol{a}^v$ in this plane.

We now present two lemmas about the neighbourhoods of vertices.

Lemma 4. (★) *Let $L = \{u \in V \mid \|\boldsymbol{a}^u\| > \sqrt{t}\}$. If $d = 2$, then $G[N(v) \cap L]$ is a co-comparability graph for all $v \in V$.*

Lemma 5. (★) *Let $u, v, w \in V$ be such that v is between u and w. If u is adjacent to w and $\|\boldsymbol{a}^v\| \geq \|\boldsymbol{a}^w\|$, then u is adjacent to v.*

We also require a result that is implied by Lemma 28 of Fiduccia et al. [10].

Lemma 6. *Suppose $d = 2$. Let u, v, and w be vertices such that v is between u and w. If u is adjacent to w, and v is adjacent to neither u nor w, then v is short.*

4 Diversity and Clustering in Social Networks

In this section, we consider the complexity of computing our two measures of diversity and clustering in social networks, i.e. Independent Set and Clique, respectively, on a dot product graph model of the network. We first prove that Independent Set is polynomial-time solvable if $d \leq 2$ and NP-complete if $d \geq 3$. We then prove that Clique is polynomial-time solvable if $d \leq 2$.

As before, throughout we have a d-dot product graph $G = (V, E)$ and a clean d-dot product representation with vectors $\{\mathbf{a}^u \mid u \in V\}$ and threshold t.

We first consider Independent Set in the case $d \leq 2$. Recall that we may assume without loss of generality that $d = 2$. Armed with the structural results of the previous section, we can prove the following theorem.

Theorem 1. Independent Set *is solvable in $O(n^3)$ time on 2-dot product graphs on n vertices.*

Proof. Let G be a 2-dot product graph. We describe how to find a maximum size independent set of G. In fact, we will describe how to find, for each long vertex u of G, the maximum size independent set of G that contains u. This is sufficient as the maximum size set of G is either the largest of these sets, or the set of all short vertices which is also independent by Lemma 3; we use this latter fact repeatedly in this proof. So let u be a fixed long vertex of G. Let G_u be the graph obtained by removing all vertices that neighbour u and their incident edges. If we can find the maximum size independent set of G_u, we will have found the maximum size independent set of G that contains u.

We define a total (or linear) ordering $\prec$ of the vertices of G_u by ordering the vertices by increasing angle of their vector representation from $\boldsymbol{a}^u$. Using the square of the cosine formula, $\prec$ can be computed in quadratic time using just dot-products. We wish to relate this ordering to betweenness. Suppose that two vertices v and w are adjacent in G_u and that θ_{vw} is positive. Any vertex between v and w is, by Lemma 6, either short or adjacent to one of them, and we know that u is a long vertex with no neighbours. So if x is between v and w, we have $v \prec x \prec w$. The converse is clearly true, giving us:

Claim 1: Let v, w, x be vertices in G_u where v and w are adjacent. Then x is between v and w and θ_{vw} is positive if and only if $v \prec x \prec w$.

For a long vertex v in G_u, let $J(v)$ be a largest independent set containing v in the subgraph of G_u that contains all vertices up to v in the ordering $\prec$, and let $j(v) = |J(v)|$. For a pair of long vertices v and w in G_u with $w \prec v$, let $S(w, v)$ be the set of vertices x such that x is short, $w \prec x \prec v$ and x is not adjacent to either v or w. Let $s(w, v) = |S(w, v)|$.

Claim 2: For each pair of non-adjacent long vertices v and w with $w \prec v$ in G_u, $j(v) \geq j(w) + s(w, v) + 1$.

Proof. Note that the claim will follow if we can show that $J(w) \cup S(w, v) \cup \{v\}$ is an independent set. All we need to show is that no vertex in $S(w, v) \cup \{v\}$ is adjacent to a vertex in $J(w)$.

Suppose that v is adjacent to a vertex x in $J(w)$. We know v and w are not adjacent so $x \neq w$ and $x \prec w \prec v$. Hence, w is between x and v (by Claim 1), and the adjacency of x and v implies, by Lemma 6, that w is short; a contradiction.

If a vertex $y \in S(w, v)$ is adjacent to any vertex x in $J(w)$, then $x \neq w$ by the definition of $S(w, v)$. But x is adjacent to w using Lemma 5 and noting that w is long, y is short and w is between x and y. This contradiction proves Claim 2.

Claim 3: For each long vertex $v \neq u$ in G_u, $j(v)$ is the maximum, over all long vertices w with $w \prec v$ and v and w non-adjacent, of $j(w) + s(w, v) + 1$.

Proof. Note that the set of long vertices that precede v includes the isolated vertex u so the maximum is well-defined, and the previous claim tells us that $j(v)$ is no less than this maximum. We must show that it is no larger. Let w be the long vertex that is last in the ordering amongst all long vertices in $J(v) \setminus \{v\}$ (as $J(v)$ contains u we can always find such a vertex). The subset of $J(v)$ containing only w and preceding vertices is independent and contains at most $j(w)$ vertices. The only other vertices in $J(v)$ are short vertices between w and v and v itself. Thus $j(v) \leq j(w) + s(w, v) + 1$, and Claim 3 is proved.

Note that j can easily be computed since $j(u) = 1$, and Claim 3 tells us that if we consider the vertices in order we can find the remaining values.

For each long vertex v in G_u, let $S^+(v)$ contain each vertex w such that w is short, $v \prec w$ and v is not adjacent to w. Let $s^+(v) = |S^+(v)|$. Let m be the maximum, over all long vertices v in G_u, of $j(v) + s^+(v)$.

Claim 4: Let J be a maximum size independent set in G_u. Then $|J| = m$.

Proof. Let v be a long vertex in G_u. We shall show that $J(v) \cup S^+(v)$ is an independent set. Let w be a vertex in $S^+(v)$ and suppose that x is a vertex in $J(v)$ adjacent to w. By the definition of $S^+(v)$, we have $x \neq v$, so $x \prec v \prec w$. By Claim 1, v is between x and w and, by Lemma 6, v is either short or adjacent to x or w. This contradiction shows that $J(v) \cup S^+(v)$ is an independent set. So $|J| \geq j(v) + s^+(v)$ for all long vertices v and hence $|J|$ is at least m.

Now let z be the long vertex in J that is latest in the ordering. Let J_1 be the subset of J containing z and preceding vertices. Hence, $|J_1| \leq j(z)$. The vertices of $J \setminus J_1$ are short vertices later than z in the ordering, so there are at most $s^+(z)$ of them. Thus $|J| \leq j(z) + s^+(z) \leq m$, and Claim 4 is proved.

We omit the details but it is straightforward to show that j and s^+, and so also m, can be computed in $O(n^2)$ time. The corresponding sets of vertices, and thus a maximum size independent set of G_u, can also be found. By repeating for each u, a maximum size independent set of G is found in time $O(n^3)$. $\square$

We contrast this positive result by the following result.

Theorem 2. (★) *For any $d \geq 3$,* Independent Set *is NP-complete on d-dot product graphs* [2].

The structural results of the previous section provide enough structure to solve Clique in polynomial time on 2-dot product graphs.

Theorem 3. (★) Clique *is solvable in $O(n^4)$ time on 2-dot product graphs on n vertices, even if no 2-dot product representation is given.*

5 Structure and Complexity for Variants of the Model

In this section, we consider two variants of the dot product graph model, which model that two individuals are connected if and only if their preferences are not opposite, or are neither opposite nor orthogonal. In the introduction, we defined the d^0-dot product graph and the d^+-dot product graph model for these cases. Recall that if $\{\mathbf{a}^u \mid u \in V\}$ is a representation of $G = (V, E)$, then

[2] Here the problem input consists of the graph, but not (necessarily) a representation.

– $(u, v) \in E$ if and only if $\mathbf{a}^u \cdot \mathbf{a}^v \geq 0$ when G is a d^0-dot product graph, and
– $(u, v) \in E$ if and only if $\mathbf{a}^u \cdot \mathbf{a}^v > 0$ when G is a d^+-dot product graph.

We study the complexity of computing the diversity and clustering measures on these models, that is, of Independent Set and Clique, on d^0-dot product graphs and d^+-dot product graphs.

Note that vertices of length 0 are adjacent to all other vertices in a d^0-dot product graph and are isolated in a d^+-dot product graph, and so do not, in either case, influence Independent Set or Clique. Hence, without loss of generality all vectors in this section have non-zero length.

First, we describe the structure of independent sets in d^0-dot product graphs. The following lemma is equivalent to Lemma 18 of Fiduccia et al. [10].

Lemma 7. *For all $d \geq 1$, every independent set in a d^0-dot product graph has size at most $d + 1$.*

Independent sets in d^+-dot product graphs have a different structure.

Lemma 8. (★) *For all $d \geq 1$, every independent set in a d^+-dot product graph has size at most $2d$.*

The proofs of Lemmas 7 and 8 can be turned into constructions to show that the given bounds are tight. The lemmas show that d^0-dot product graphs and d^+-dot product graphs have different structure, which is also different from the structure of d-dot product graphs. Moreover, using exhaustive enumeration, the two lemmas immediately imply the following.

Theorem 4. *For all $d \geq 1$,* Independent Set *is solvable in $O(n^{d+1})$ time on d^0-dot product graphs and in $O(n^{2d})$ time on d^+-dot product graphs on n vertices, even if no representation is given.*

We now consider Clique on d^0-dot product and d^+-dot product graphs. For $d = 2$, it suffices to observe that a set of vertices forms a clique if and only if their corresponding vectors lie in the nonnegative quadrant (after an appropriate rotation). However, this structural observation does not generalize to higher dimensions, as is evident from the counterexamples by Gray and Wilson [12] for $d = 3$ and $d \geq 5$. Instead, we follow a different approach, which leads to a polynomial-time algorithm for all $d \leq 3$.

For any hyperplane h with normal $\mathbf{n}$, let h^+ be the half-space $\{p \mid p \cdot \mathbf{n} \geq 0\}$ and let h^- be the half-space $\{p \mid p \cdot \mathbf{n} \leq 0\}$. Note that any two vectors $\mathbf{a}, \mathbf{b}$ induce a hyperplane with normal $\mathbf{a} \times \mathbf{b}$, where $\times$ is the cross product operation. We refer to the monograph by Barvinok [4] for any undefined terminology on cones.

Theorem 5. *For all $d \leq 3$,* Clique *can be solved in $O(n^{4.5})$ time on d^0-dot product graphs and d^+-dot product graphs on n vertices.*

Proof. We assume that $d = 3$ (fewer dimensions are a special case). Let $G = (V, E)$ be a 3^0-dot product graph or a 3^+-dot product graph with representation

$\{\mathbf{a}^v \mid v \in V\}$. We first give a structural result, where we essentially show that any clique C of G induces a basis such that the vectors of C lie in two octants with respect to this basis. Then, we give an algorithm that finds this basis for a maximum clique by guessing limited information about the clique, and uses the basis to obtain a maximum clique of G.

We start with the structural result. Let C be any clique of G. Let $\mathcal{K}$ denote the conic hull of $\mathbf{a}^v$ for all vertices $v \in C$, that is, $\mathcal{K} = \{\sum_{v \in C} \lambda_v \mathbf{a}^v \mid \lambda_v \geq 0\}$. We call $\mathcal{K}$ the *cone corresponding to* C. The structural result considers the case that $\mathcal{K}$ is not a ray (a *ray* is the conic hull of a single vector). Since $\mathcal{K}$ is generated by a finite set, its extreme rays are vectors that correspond to vertices of C. Let u be any vertex such that $\mathbf{a}^u$ spans an extreme ray of $\mathcal{K}$, and let h_u denote the hyperplane with normal $\mathbf{a}^u$. Because $\mathcal{K}$ is the conic hull of vectors corresponding to a clique, $\mathbf{p} \cdot \mathbf{a}^u \geq 0$ for any $\mathbf{p} \in \mathcal{K}$ (this is true both when G is a 3^0-dot product graph or a 3^+-dot product graph). Hence, $\mathcal{K} \subseteq h_u^+$.

Let w be any vertex such that $\mathbf{a}^w$ spans an extreme ray of $\mathcal{K}$ that is not spanned by u and such that the hyperplane h_{uw} induced by $\mathbf{a}^u$ and $\mathbf{a}^w$ contains a facet of $\mathcal{K}$. Since h_{uw} contains a facet of $\mathcal{K}$, either $\mathcal{K} \subseteq h_{uw}^+$ or $\mathcal{K} \subseteq h_{uw}^-$. Assume without loss of generality that $\mathcal{K} \subseteq h_{uw}^+$, and let $\mathbf{t}$ denote the normal of h_{uw} that lies in h_{uw}^+. Finally, let $\mathbf{w}'$ denote the projection of $\mathbf{a}^w$ onto h_u. By definition, $\mathbf{t}$, $\mathbf{a}^u$, $\mathbf{w}'$ are pairwise orthogonal. Moreover, as $\mathcal{K} \subseteq h_u^+ \cap h_{uw}^+$ and $h_u^+ \cap h_{uw}^+$ is the union of two octants in the basis induced by $\mathbf{t}$,$\mathbf{a}^u$,$\mathbf{w}'$, we find that $\mathcal{K}$ is a subset of two octants in the basis induced by $\mathbf{t}$, $\mathbf{a}^u$, $\mathbf{w}'$.

We use the structural result in an algorithm that consists of two phases.

In the first phase of the algorithm, we ensure that we find a maximum clique if the cone corresponding to some maximum clique is a ray. Therefore, we iterate over all $v \in V(G)$ and find the set X of vertices u for which $\mathbf{a}^u$ spans the same ray as $\mathbf{a}^v$. The set X is a clique irrespective of whether G is a 3^0-dot product graph or a 3^+-dot product graph. We keep a maximum clique found over all choices of v.

In the second phase of the algorithm, we ensure that we find a maximum clique if the cone corresponding to some maximum clique is not a ray. Iterate over all n^2 ordered pairs (u, w) of the vertices of G such that $\mathbf{a}^u$ and $\mathbf{a}^w$ do not span the same ray. Define h_u as the plane with normal $\mathbf{a}^u$, and define h_{uw} as the plane induced by $\mathbf{a}^u$ and $\mathbf{a}^w$. Consider $h_u^+ \cap h_{uw}^+$ (we also consider $h_u^+ \cap h_{uw}^-$ in a similar way). Let $\mathbf{t}$ denote the normal of h_{uw} that lies in h_{uw}^+ and let $\mathbf{w}'$ denote the projection of $\mathbf{a}^w$ onto h_w. Note that $h_u^+ \cap h_{uw}^+$ is the union of two octants in the basis induced by $\mathbf{t}$, $\mathbf{a}^u$, $\mathbf{w}'$. As any octant induces a clique, $h_u^+ \cap h_{uw}^+$ induces a co-bipartite graph H. We can find H in linear time as the graph induced by the vertices whose corresponding vectors have positive or strictly positive dot product with both $\mathbf{a}^u$ and $\mathbf{t}$. Since H is co-bipartite, we can find a maximum clique of H in $O(n^{2.5})$ time, as it reduces to finding a maximum matching in a bipartite graph, which takes $O(n^{2.5})$ time [15]. We then keep a maximum clique over all choices of u, w. The output of the algorithm is a largest of the two cliques kept in the first and second phase.

The algorithm runs in $O(n^{4.5})$ time, as claimed. To see correctness, let C be a maximum clique. If the cone corresponding to C is a ray, then the algorithm considers C in the first phase. If the cone corresponding to C is not a ray, then by our structural result there will be a choice of u, w for which $u, w \in C$ and h_{uw} contains a facet of $\mathcal{K}$, where $\mathcal{K}$ is the cone corresponding to C. $\square$

6 Conclusions

This paper provided the first study of algorithms that measure diversity and clustering in social networks that are modeled as dot product graphs. The diversity and clustering measures considered correspond to Independent Set and Clique on dot product graphs.

Our exploration of the complexity of Clique on d-dot product graphs leaves further open problems. The current approach for $d = 2$ does not seem to extend to d-dot product graphs for $d \geq 3$, as our structural results (Lemma 2 for example) seem to indicate that we need to solve clique on co-p-partite graphs for $p \geq 3$. However, this problem is NP-complete, as Independent Set is NP-complete on 2-subdivisions of planar graphs [11]. Hence, further structural insight into d-dot product graphs is needed to resolve the complexity of Clique on these graphs.

We observe that our polynomial-time algorithms for Independent Set and Clique on 2-dot product graphs generalize well-known polynomial-time algorithms for these problems on interval graphs, because interval graphs have a 2-dot product representation [10, Theorem 21]. At the same time, we are unaware of any nontrivial superclasses of 2-dot product graphs, in particular for which Independent Set and Clique are polynomial-time solvable. Finally, we note that the dot product graph model of social networks might be able to capture more problems for social networks as graph optimization problems.

References

1. Adamic, L.A., Adar, E.: Friends and neighbors on the Web. Social Networks 25, 211–230 (2003)
2. Arora, S., Steurer, D., Wigderson, A.: Towards a Study of Low-Complexity Graphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 119–131. Springer, Heidelberg (2009)
3. Barabási, A.-L., Albert, R.: Emergence of Scaling in Random Networks. Science 286, 509–512 (1999)
4. Barvinok, A.: A Course in Convexity. American Mathematical Society (2003)
5. Breiger, R.L.: The Duality of Persons and Groups. Social Forces 53, 181–190 (1974)
6. Bhawalkar, K., Kleinberg, J., Lewi, K., Roughgarden, T., Sharma, A.: Preventing Unraveling in Social Networks: The Anchored k-Core Problem. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 440–451. Springer, Heidelberg (2012)
7. Caldarelli, G., Capocci, A., de Los Rios, P., Muñoz, M.A.: Scale-Free Networks from Varying Vertex Intrinsic Fitness. Phys. Rev. Lett. 89, 258702 (2002)
8. Diestel, R.: Graph Theory. Springer (2005)
9. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for W[1]. Theoretical Computer Science 141, 109–131 (1995)
10. Fiduccia, C.M., Scheinerman, E.R., Trenk, A., Zito, J.S.: Dot product representations of graphs. Discrete Mathematics 181, 113–138 (1998)

11. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comp. Sci. 1, 237–267 (1976)
12. Gray, L.J., Wilson, D.G.: Nonnegative factorization of positive semidefinite nonnegative matrices. Linear Algebra and its Applications 31, 119–127 (1980)
13. Håstad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Mathematica 182, 105–142 (1999)
14. Hoff, P.D., Raftery, A.E., Handcock, M.S.: Latent Space Approaches to Social Network Analysis. J. Am. Stat. Assoc. 97, 1090–1098 (2002)
15. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. 2, 225–231 (1973)
16. Kang, R.J., Lovász, L., Müller, T., Scheinerman, E.R.: Dot product representations of planar graphs. Electr. J. Comb. 18 (2011)
17. Kang, R.J., Müller, T.: Sphere and dot product representations of graphs. Discrete and Computational Geometry 47, 548–568 (2012)
18. Karp, R.M.: Reducibility among Combinatorial Problems, Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
19. Kim, M., Leskovec, J.: Multiplicative Attribute Graph Model of Real-World Networks. In: Kumar, R., Sivakumar, D. (eds.) WAW 2010. LNCS, vol. 6516, pp. 62–73. Springer, Heidelberg (2010)
20. Kotlov, A., Lovász, L., Vempala, S.: The Colin de Verdiére number and sphere representations of a graph. Combinatorica 17, 483–521 (1997)
21. Liben-Nowell, D., Kleinberg, J.: The Link Prediction Problem for Social Networks. In: Proc. CIKM 2003, pp. 556–559 (2003)
22. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a Feather: Homophily in Social Networks. Annual Rev. Sociology 27, 415–444 (2001)
23. Newman, M.E.J.: The Structure and Function of Complex Networks. SIAM Review 45, 167–256 (2003)
24. Nickel, C.M.L.: Random Dot Product Graphs: A Model for Social Networks. PhD dissertation, Johns Hopkins University (2007)
25. Papadimitriou, C.H., Raghavan, P., Tamaki, H., Vempala, S.: Latent Semantic Indexing: A Probabilistic Analysis. J. Comput. Syst. Sci. 61, 217–235 (2000)
26. de, D.J., Price, S.: A general theory of bibliometric and other cumulative advantage processes. J. American Society for Information Science 27, 292–306 (1976)
27. Reiterman, J., Rödl, V., Šinăjová, E.: Embeddings of graphs in Euclidean spaces. Discrete and Computational Geometery 4, 349–364 (1989)
28. Reiterman, J., Rödl, V., Šinăjová, E.: Geometrical embeddings of graphs. Discrete Mathematics 74, 291–319 (1989)
29. Reiterman, J., Rödl, V., Šinăjová, E.: On embedding of graphs into Euclidean spaces of small dimension. J. Combin. Theory B 56, 1–8 (1992)
30. Scheinerman, E.R., Tucker, K.: Modeling graphs using dot product representations. Computational Statistics 25, 1–16 (2010)
31. Simon, H.A.: On a class of skew distribution functions. Biom. 42, 425–440 (1955)
32. Snijders, T.A.B.: Statistical Models for Social Networks. Annual Rev. Sociology 37, 131–153 (2011)
33. Watts, D.J., Dodds, P.S., Newman, M.E.J.: Identity and Search in Social Networks. Science 296, 1302–1305 (2002)
34. Young, S.J., Scheinerman, E.R.: Random dot product graph models for social networks. In: Bonato, A., Chung, F.R.K. (eds.) WAW 2007. LNCS, vol. 4863, pp. 138–149. Springer, Heidelberg (2007)
35. Young, S.J., Scheinerman, E.R.: Directed random dot product graphs. Internet Mathematics 5, 91–111 (2008)

Sublinear-Time Algorithms for Monomer-Dimer Systems on Bounded Degree Graphs [⋆]

Marc Lelarge and Hang Zhou

INRIA, École Normale Supérieure, France
{marc.lelarge,hang.zhou}@ens.fr

Abstract. For a graph G, let $Z(G,\lambda)$ be the partition function of the monomer-dimer system defined by $\sum_k m_k(G)\lambda^k$, where $m_k(G)$ is the number of matchings of size k in G. We consider graphs of bounded degree and develop a sublinear-time algorithm for estimating $\log Z(G,\lambda)$ at an arbitrary value $\lambda > 0$ within additive error ϵn with high probability. The query complexity of our algorithm does not depend on the size of G and is polynomial in $1/\epsilon$, and we also provide a lower bound quadratic in $1/\epsilon$ for this problem. This is the first analysis of a sublinear-time approximation algorithm for a $\#P$-complete problem. Our approach is based on the correlation decay of the Gibbs distribution associated with $Z(G,\lambda)$. We show that our algorithm approximates the probability for a vertex to be covered by a matching, sampled according to this Gibbs distribution, in a near-optimal sublinear time. We extend our results to approximate the average size and the entropy of such a matching within an additive error with high probability, where again the query complexity is polynomial in $1/\epsilon$ and the lower bound is quadratic in $1/\epsilon$. Our algorithms are simple to implement and of practical use when dealing with massive datasets. Our results extend to other systems where the correlation decay is known to hold as for the independent set problem up to the critical activity.

1 Introduction

The area of sublinear-time algorithms is an emerging area of computer science which has its root in the study of massive data sets [6,24]. Internet, social networks or communication networks are typical examples of graphs with potentially millions of vertices representing agents, and edges representing possible interactions among those agents. In this paper, we present sublinear-time algorithms for graph problems. We are concerned more with problems of counting and statistical inference and less with optimization. For example, in a mobile call graphs, phone calls can be represented as a matching of the graph where each edge has an activity associated to the intensity of the interactions between the pair of users. Given such a graphs, with local activities on edges, we would like to answer questions like: what is the size of a typical matching? for a given user what is the probability of being matched? As another example, models of

[⋆] Full version available at http://arxiv.org/abs/1208.3629 statistical physics have been proposed to model social interactions. In particular, spin systems are a general framework for modeling nearest-neighbor interactions on graphs. In this setting, the activity associated to each edge allows to model a perturbed best-response dynamics [2]. Again in this setting, it is interesting to compute estimations for the number of agents playing a given strategy or the probability for an agent in the graph to play a given strategy at equilibrium.

There are now quite a few results on sublinear-time approximation algorithms for graph optimization problems: minimum spanning tree weight [5], minimum set cover [21], maximum matching [21,32] and minimum vertex cover [21,22,23]. There are also a couple of works on sublinear-time algorithms for statistical and counting problems, e.g., approximating the average degree of a graph [7,11] and approximating the number of occurrences of a certain structure (such as a star) in a graph [12]. Our focus in this paper is on the algorithmic problems arising in statistical physics and classical combinatorics [30]. We now present the monomer-dimer problem which will be the main focus of our paper.

Let $G = (V, E)$ be an undirected graph with $|V| = n$ vertices and $|E| = m$ edges, where we allow G to contain parallel edges and self-loops. We denote by $N(G, v)$ the set of neighbors of v in G. We consider bounded degree graphs with $\max_v |N(G, v)| \leq \Delta$. In a monomer-dimer system, the vertices are covered by non-overlapping arrangement of monomers (molecules occupying one vertex of G) and dimers (molecules occupying two adjacent vertices of G) [13]. It is convenient to identify monomer-dimer arrangements with matchings; a matching in G is a subset $M \subset E$ such that no two edges in M share an endpoint. Thus, a matching of cardinality $|M| = k$ corresponds exactly to a monomer-dimer arrangement with k dimers and $n-2k$ monomers. Let $\mathbb{M}$ be the set of matchings of G. To each matching M, a weight $\lambda^{|M|}$ is assigned, where $\lambda > 0$ is called the activity. The partition function of the system is defined by $Z(G, \lambda) = \sum_{M \in \mathbb{M}} \lambda^{|M|}$, and the Gibbs distribution on the space $\mathbb{M}$ is defined by $\pi_{G,\lambda}(M) = \frac{\lambda^{|M|}}{Z(G,\lambda)}$. The function $Z(G, \lambda)$ is also of combinatorial interest and called the *matching polynomial* in this context [19]. For example, $Z(G, 1)$ enumerates all matchings in G. From an algorithmic viewpoint, no feasible method is known for computing $Z(G, \lambda)$ exactly for general monomer-dimes system; indeed, for any fixed value of $\lambda > 0$, the problem of computing $Z(G, \lambda)$ exactly in a graph of bounded degree Δ is complete for the class #P of enumeration problems, when $\Delta \geq 5$ (see [28]). The focus on computing $Z(G, \lambda)$ shifted to finding approximate solutions in polynomial time. For example, the Markov Chain Monte Carlo (MCMC) method yields a provably efficient algorithm for finding an approximate solution. Based on the equivalence between the counting problem (computing $Z(G, \lambda)$) and the sampling problem (according to $\pi_{G,\lambda}$) [16], this approach focuses on rapidly mixing Markov chains to obtain appropriate random samples. A Fully Polynomial-time Randomized Approximation Scheme (FPRAS) for computing the total number of matchings based on MCMC was provided by Jerrum and Sinclair [14,25].

Another related problem in the monomer-dimer system is the average size of a matching sampled according to $\pi_{G,\lambda}$, defined by $E(G, \lambda) = \sum_{M \in \mathbb{M}} |M| \, \pi_{G,\lambda}(M)$. Sinclair and Srivastava recently proved in [27] that for any fixed value of $\lambda > 0$, the problem of computing $E(G, \lambda)$ exactly in a bounded degree graph (allowing parallel edges) is #P-hard, for any maximum degree $\Delta \geq 5$. Thus again we are interested in finding approximate solutions to this problem.

In order to study sublinear-time approximation algorithms for these problems, we use the approach based on the concept of correlation decay originating in statistical physics [20] and which has been used to get a deterministic approximation scheme for counting matchings in polynomial time [1]. It follows already from [13] that the marginals of the probability distribution $\pi_{G,\lambda}$ are local in nature: the local structure of the graph around a vertex v allows to compute an approximation of the corresponding marginal. In the computer science literature, this property follows from the so-called *correlation decay property.* Our algorithm is then simple to understand: we need only to sample a fixed number of vertices, approximate the marginals associated to these vertices locally and then from these values output an estimate for the desired quantity. The correlation decay property also holds for other systems such as the independent set problem [29], the coloring problem [9], and the two-state spin system [17,18,26]. In the full version of the paper, we extend our technique to the independent set problem. We believe that similar extensions can be done for other systems as soon as the correlation decay property holds.

A graph G is represented by two kinds of oracles $\mathcal{D}$ and $\mathcal{N}$ such that $\mathcal{D}(v)$ returns the degree of $v \in V$ and $\mathcal{N}(v, i)$ returns the i^{th} (with $1 \leq i \leq \mathcal{D}(v)$) neighbor of $v \in V$. The efficiency of an algorithm is measured by its query complexity, i.e. the total number of accesses to $\mathcal{D}$ and $\mathcal{N}$. Let VAL denote a real value associated with the graph. We say that $\widehat{\text{VAL}}$ is an *ϵ-approximation* of VAL if $\widehat{\text{VAL}} - \epsilon \leq \text{VAL} \leq \widehat{\text{VAL}} + \epsilon$, where $\epsilon > 0$ is specified as an input parameter. An algorithm is called an *ϵ-approximation algorithm* for VAL if for any graph G, it computes an ϵ-approximation of VAL with high probability (e.g., at least $\frac{2}{3}$). In our model, we consider the case of constant maximum degree Δ as ϵ tends to zero, i.e., we always first take the limit as $\epsilon \to 0$ and then the limit $\Delta \to \infty$.

Our main contribution (Theorem 5) is an ϵn-approximation algorithm for $\log Z(G, \lambda)$ in a graph G of bounded degree Δ. The query complexity of the algorithm is $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$, which does not depend on the size of the graph. From the relation between the partition function and the matching statistics, we then obtain ϵn-approximation algorithms for the average size of a matching and the entropy of $\pi_{G,\lambda}$ with the same query complexity as before. We also provide the $\Omega(1/\epsilon^2)$ query lower bound for ϵn-approximation algorithms for $\log Z(G, \lambda)$ and the other two problems.

The main tool of the above algorithms is the approximation of the marginal $p_{G,\lambda}(v)$, which is the probability that the vertex v is not covered by a matching under the Gibbs distribution. We estimate $p_{G,\lambda}(v)$ for an arbitrary vertex $v \in V$ within an error of $\epsilon > 0$ with near-optimal query complexity $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$.

The rest of the paper is organized as follows. In Section 2, we prove our first main result concerning local computations for matchings. Based on this result, we construct an ϵn-approximation algorithm for the partition function $Z(G, \lambda)$

in Section 3 and ϵn-approximation algorithms for the average size of a matching and the entropy of $\pi_{G,\lambda}$ in Section 4. We also provide query lower bounds in these two sections. In Section 5, we give some applications of our technique for approximating the permanent of constant degree expander graphs and the size of a maximum matching (in this last case, our algorithm is outperformed by [32]). In the full version of the paper, we also show the efficiency of our algorithms by testing on large real-world networks.

2 Local Computations for Matchings

Recall that we defined for all $\lambda > 0$, the Gibbs distribution on matchings of a graph G by:

$$\forall M \in \mathbb{M}, \quad \pi_{G,\lambda}(M) = \frac{\lambda^{|M|}}{Z(G,\lambda)} \text{ where } Z(G,\lambda) = \sum_{M \in \mathbb{M}} \lambda^{|M|}.$$

The focus in this section is on the approximation of the probability that a vertex $v \in V$ is not coverd by a matching:

$$p_{G,\lambda}(v) := \sum_{M \not\ni v} \pi_{G,\lambda}(M),$$

where $M \not\ni v$ is a matching not covering v.

First notice that

$$p_{G,\lambda}(v) = \frac{Z(G\backslash\{v\},\lambda)}{Z(G,\lambda)}, \tag{1}$$

where $G\backslash\{v\}$ is the graph obtained from G by removing the vertex v and all incident edges. Then we have

$$Z(G,\lambda) = Z(G\backslash\{v\},\lambda) + \lambda \sum_{u \in N(G,v)} Z(G\backslash\{u,v\},\lambda),$$

so that dividing by $Z(G\backslash\{v\},\lambda)$, we get

$$p_{G,\lambda}(v) = \frac{1}{1 + \lambda \sum_{u \in N(G,v)} p_{G\backslash\{v\},\lambda}(u)}. \tag{2}$$

This recursive expression for $p_{G,\lambda}(v)$ is well-known and allows to compute the marginal $p_{G,\lambda}(v)$ exactly for each $v \in V$. We follow the approach of Godsil [10]. First, we recall the notion of *path-tree* associated with a rooted graph: if G is any rooted graph with root v_0, we define its path-tree $T_G(v_0)$ as the rooted tree whose vertex-set consists of all *finite simple paths* starting at the root v_0; whose edges are the pairs $\{P, P'\}$ of the form $P = v_0 \dots v_k$, $P' = v_0 \dots v_k v_{k+1} (k \geq 0)$; and whose root is the single-vertex path v_0. By a *finite simple path*, we mean here a finite sequence of distinct vertices $v_0 \dots v_k$ $(k \geq 0)$ such that $v_i v_{i+1} \in E$ for all $0 \leq i < k$. Note that the notion of path-tree is similar to the more standard notion of computation tree, the main difference being that for any finite graph G, the path-tree is always finite (although its size might be much larger than the size of the original graph G).

For every node u in the path-tree $T_G(v)$, define $\mathsf{Ch}(u)$ to be the set of children of u in $T_G(v)$. The recursion (2) easily implies $p_{G,\lambda}(v) = p_{T_G(v),\lambda}(v)$ and $p_{T_G(v),\lambda}(v) = x_v(v)$, where the vector $\mathbf{x}(v) = (x_u(v),\ u \in T_G(v))$ solves the recursion:

$$\forall u \in T_G(v), \quad x_u(v) = \frac{1}{1 + \lambda \sum_{w \in \mathsf{Ch}(u)} x_w(v)} \tag{3}$$

(by convention a sum over the empty set is zero).

In order to approximate $p_{G,\lambda}(v)$, we will show that it suffices to solve the recursion (3) restricted to a truncated path-tree of $T_G(v)$. For any $h \geq 1$, let $T_G^h(v)$ be the path-tree truncated at depth h and let $\mathbf{x}^h(v) = (x_u^h(v),\ u \in T_G^h(v))$ be the solution of the recursion (3) when the path-tree is replaced by the truncated version $T_G^h(v)$. Clearly $x_v^h(v) = p_{G,\lambda}(v)$ for any $h \geq n$ and the following lemma gives a quantitative estimate on how large h needs to be in order to get an ϵ-approximation of $p_{G,\lambda}(v)$.

Lemma 1. *There exists $\overline{h}(\epsilon, \Delta)$ such that $|\log x_v^h(v) - \log p_{G,\lambda}(v)| \leq \epsilon$ for any $h \geq \overline{h}(\epsilon, \Delta)$. Moreover $\overline{h}(\epsilon, \Delta) = \tilde{O}\left(\sqrt{\Delta} \log(1/\epsilon)\right)$ and satisfies*

$$\lim_{\Delta \to \infty} \frac{1}{\sqrt{\Delta}} \lim_{\epsilon \to 0} \frac{\overline{h}(\epsilon, \Delta)}{\log(1/\epsilon)} = \sqrt{\lambda}.$$

Proof. Theorem 3.2 in [1] proves that:

$$|\log x_v^h(v) - \log p_{G,\lambda}(v)| \leq \left(1 - \frac{2}{\sqrt{1 + \lambda\Delta} + 1}\right)^{h/2} \log(1 + \lambda\Delta). \tag{4}$$

The lemma then follows directly by taking $\overline{h}(\epsilon, \Delta)$ to be the h such that the right-hand side equals ϵ. □

We now present the algorithmic implication of Lemma 1. We start with a simple remark. The exact value for $\overline{h}(\epsilon, \Delta)$ follows from the proof of the lemma, however this value will not be required in what follows as shown by the following argument: the fact that $(z_1, \ldots, z_\Delta) \mapsto \left(1 + \lambda \sum_{i=1}^{\Delta} z_i\right)^{-1}$ is strictly decreasing in each positive variable z_i implies (by a simple induction) that for any $k \geq 0$, we have

$$x_v^{2k+1}(v) \leq x_v^{2k+3}(v) \leq p_{G,\lambda}(v) \leq x_v^{2k+2}(v) \leq x_v^{2k}(v). \tag{5}$$

Consider an algorithm that computes $x_v^h(v)$ for increasing values of h and stops at the first time two consecutive outputs are such that $|\log x_v^{h+1}(v) - \log x_v^h(v)| \leq \epsilon$. By Lemma 1, it takes at most $\overline{h}(\epsilon, \Delta)$ iterations and the last output will be an ϵ-approximation of $\log p_{G,\lambda}(v)$.

The algorithm Approx-Marginal(λ, ϵ, v) provides an estimate of $p_{G,\lambda}(v)$, based on the Depth-First-Search (DFS) on the truncated path-tree rooted at v. In the algorithm DFS(λ, h, s, ℓ), integer h is the truncated level of the path tree $T_G(v)$; $s \in V$ is the current node in the graph G visiting by the DFS; and *path* maintains an array of nodes in G which form the path from v to s during the DFS. This path also corresponds to a node in the path-tree $T_G^h(v)$ and let ℓ be the length of *path*. The algorithm DFS(λ, h, s, ℓ) computes recursively the marginal probability of *path* in $T_G^h(v)$. Recall that $\mathcal{D}(v)$ returns the degree of $v \in V$ and $\mathcal{N}(v, i)$ returns the i^{th} (with $1 \leq i \leq \mathcal{D}(v)$) neighbor of $v \in V$.

```
Approx-Marginal($\lambda, \epsilon, v$)
1  $x[1] \leftarrow$ DFS($\lambda, 1, 1, v$)
2  $x[2] \leftarrow$ DFS($\lambda, 1, 2, v$)
3  $h \leftarrow 2$
4  **while** $|\log x[h] - \log x[h-1]| > \epsilon/e$
5     **do** $h \leftarrow h + 1$
6          $x[h] \leftarrow$ DFS($\lambda, h, v, 1$)
7  **return** $x[h]$
```

```
DFS($\lambda, h, s, \ell$)
1  **if** $\ell = h$
2     **then return** 1
3  $A \leftarrow 1,\ path[\ell] \leftarrow s$
4  **for** $i \leftarrow 1$ **to** $\mathcal{D}(s)$
5     **do** $t \leftarrow \mathcal{N}(s, i)$
6          **if** $\forall j \in [1, \ell],\ t \neq path[j]$
7             **then** $A \leftarrow A +$ DFS($\lambda, h, t, \ell + 1$)
8  **return** $1/(\lambda A)$
```

Proposition 2. *The algorithm* Approx-Marginal*(λ, ϵ, v) gives an estimate $\widehat{p}$ of $p_{G,\lambda}(v)$, such that $|\widehat{p} - p_{G,\lambda}(v)|$ and $|\log \widehat{p} - \log p_{G,\lambda}(v)|$ are both smaller than ϵ. Its query complexity is $\overline{\mathcal{Q}}(\epsilon, \Delta) = \tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$. In addition, $Q = \overline{\mathcal{Q}}(\epsilon, \Delta)$ satisfies*

$$\lim_{\Delta \to \infty} \frac{1}{\sqrt{\Delta} \log \Delta} \lim_{\epsilon \to 0} \frac{\log Q}{\log(1/\epsilon)} = \sqrt{\lambda}. \tag{6}$$

Proof. Let h be the final truncated level of the path-tree $T_G(v)$ in the algorithm. We have $\widehat{p} = x_v^h(v)$ and $|\log x_v^h(v) - \log x_v^{h-1}(v)| < \epsilon/e$. Thus $|\log \widehat{p} - \log p_{G,\lambda}(v)| < \epsilon/e$ by Inequality (5). Since $p_{G,\lambda}(v)$ and $\widehat{p}$ are at most 1, we then have $|\widehat{p} - p_{G,\lambda}(v)| < \epsilon$. The number of nodes visited by the algorithm is $O\left(\Delta^h\right)$, so the number of queries is also $O\left(\Delta^h\right)$. The proposition follows by applying the upper bound $\overline{h}(\epsilon, \Delta)$ on h from Lemma 1. □

Remark. *In Section 3, we need to estimate the marginal probability at the node v in the graph $G_v = \{u \in V \mid u \succeq v\}$ instead of the graph G, where $\succ$ is some* *total order over V. To achieve this, we only need to add an additional constraint $t \succeq v$ to line 6 of the DFS algorithm. Denote* Approx-Marginal$^*(\lambda, \epsilon, v)$ *to be the modified version of* Approx-Marginal(λ, ϵ, v) *with the underlying graph G_v. Again, Proposition 2 holds for the algorithm* Approx-Marginal$^*(\lambda, \epsilon, v)$ *by replacing G by G_v.*

The next propostion shows that there exists some $\underline{Q}(\epsilon, \Delta)$, such that $Q = \underline{Q}(\epsilon, \Delta)$ satisfies Equation (6) and that $\underline{Q}(\epsilon, \Delta)$ is a query lower bound for computing an ϵ-approximation of $p_{G,\lambda}(v)$. This implies that Algorithm Approx-Marginal is optimal when the influence of ϵ is much larger than that of Δ. The idea of the lower bound proof is to construct two instances of *almost full Δ-ary trees* whose marginal probabilities at the root differ by more than ϵ, while any approximation algorithm using a small number of queries cannot distinguish them. See the full version of the paper for a detailed proof of this proposition.

Proposition 3. *In order to approximate the marginal $p_{G,\lambda}(v)$ within an additive error ϵ, any deterministic or randomized algorithm*[1] *requires $\Omega\left(\underline{Q}(\epsilon, \Delta)\right)$ queries where $Q = \underline{Q}(\epsilon, \Delta)$ satisfies Equation* (6).

Remark. *As noted in the introduction, the model with λ_e ($e \in E$) varying across the edges is of practical interest (allowing to model various intensities on edges). As soon as there exists $\lambda_{\max}$ such that for all $e \in E$, we have $\lambda_e \in [0, \lambda_{\max}]$, it is easy to extend the results of this section to the more general model defined by (note that $\boldsymbol{\lambda}$ is now a vector in $[0, \lambda_{\max}]^E$): $\pi_{G,\boldsymbol{\lambda}}(M) = \frac{\prod_{e\in M} \lambda_e}{Z(G,\boldsymbol{\lambda})}$ where, $Z(G, \boldsymbol{\lambda}) = \sum_{M\in\mathbb{M}} \prod_{e\in M} \lambda_e$. Results in this section and Sections 3 and 4 hold provided λ is replaced by $\lambda_{\max}$.*

3 Approximating the Partition Function

First, we need an arbitrary total order $\succ$ over V. We can achieve this by assigning a random number $a_v \in [0, 1]$ to each vertex v and then defining $u \succ v$ as $a_u > a_v$. However, if there are only a small number of vertices involved in our computation, we do not need to generate random numbers for all vertices. Using the technique in [22], we generate a random number each time we visit a new vertex and then save its value for the later visits. Generating a random number can be done in sublinear time and the number of vertices in our computation is at most twice the number of queries, which will later be proved to be a constant independent of n. As a result, the total time complexity for this random generation is sublinear.

Define $G_v = \{u \in V \mid u \succeq v\}$. The following formula which allows us to compute the partition function from the marginals is obtained easily from (1):

$$\log Z(G, \lambda) = \sum_{v\in V} - \log p_{G_v,\lambda}(v). \tag{7}$$

[1] In the randomized case, the algorithm is expected to provide *always* an estimate with an additive error ϵ, and the proposition implies a lower bound on the *average* number of queries of such an algorithm.

The algorithm below estimates $\log Z(G, \lambda)$. We sample $\lceil C/\epsilon^2 \rceil$ vertices uniformly at random from V, where C is some fixed constant. For every sampled vertex u, we compute an estimate of the marginal $p_{G_u,\lambda}(u)$ using the algorithm Approx-Marginal*$(\lambda, \epsilon/2, u)$. We then obtain an estimate of $Z(G, \lambda)$ from the estimates of marginals at sampled vertices.

```
Approx-Partition-Function($\lambda, \epsilon$)
1  $s \leftarrow \lceil C/\epsilon^2 \rceil$
2  $U \leftarrow$ random multi-subset of $V$ with $s$ elements
3  return $(n/s) \cdot \left(\sum_{u \in U} -\log(\text{Approx-Marginal}^*(\lambda, \epsilon/2, u))\right)$
```

We recall a basic lemma which follows from Hoeffding's inequality and which will be used several times in the paper.

Lemma 4. *(see [4]) Let V be a set of n real numbers in $[A, B]$, where A and B are constant. Let V' be a multi-subset of V consisting of $\Theta(1/\epsilon^2)$ elements chosen uniformly and independently at random. Let* AVG *be the average of all elements and* AVG$'$ *be the average of sampled elements. Then with high constant probability, we have:* $\text{AVG}' - \epsilon \leq \text{AVG} \leq \text{AVG}' + \epsilon$.

Theorem 5. Approx-Partition-Function*(λ, ϵ) is an ϵn-approximation algorithm for* $\log Z(G, \lambda)$ *with query complexity* $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$.

Proof. Let $A = \sum_{v \in V} -\log(\text{Approx-Marginal}^*(\lambda, \epsilon/2, v))$. By Proposition 2 and Equation (7), A is an $\epsilon n/2$-approximation of $\log Z(G, \lambda)$. By Lemma 4, there exists some constant C such that approximating the marginal probability at $\lceil C/\epsilon^2 \rceil$ sampled nodes gives an $\epsilon n/2$-approximation of A with high probability. This implies an ϵn-approximation of $\log Z(G, \lambda)$ with high probability. The query complexity of this algorithm is $\lceil C/\epsilon^2 \rceil \cdot \overline{\mathcal{Q}}(\epsilon/2, \Delta) = \tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$. □

Note that the size of any maximal matching is always lower bounded by $\frac{m}{2\Delta-1}$, where m is the number of edges. In particular, since $Z(G, 1)$ is the total number of matchings, we have $\frac{m}{2\Delta-1} \log 2 \leq \log Z(G, 1) \leq m \log 2 \leq \frac{n\Delta}{2} \log 2$ so that if $m = \Omega(n)$, we also have $\log Z(G, 1) = \Theta(n)$. Hence, if ϵ and Δ are constants and $m = \Omega(n)$, the error in the output of our algorithm is of the same order as the evaluated quantity. This is in contrast with the FPTAS (Fully Polynomial-Time Approximation Scheme) in [1] or the FPRAS (Fully Polynomial-time Randomized Approximation Scheme) in [14,25] which outputs an ϵ-approximation instead of an ϵn-approximation. Of course, we can let ϵ tend to 0 with n like c/n in Theorem 5, so that our result (when Δ is constant) is consistent with the FPTAS in [1]. Indeed, in this case, clearly no sampling is required and if we replace the sampling step by a visit of each vertex, our algorithm is the same as in [1].

When we assume Δ to be fixed, the query complexity of the above algorithm is polynomial in $1/\epsilon$. Next we give a lower bound on the query complexity which is quadratic in $1/\epsilon$. In the proof, we use a lower bound result from [5], which is based on Yao's Minimax Principle [31]. See the full version of the paper for the proof of the following theorem.

Theorem 6. *Any deterministic or probabilistic ϵn-approximation algorithm for $\log Z(G,\lambda)$ needs $\Omega(1/\epsilon^2)$ queries on average. It is assumed that $\epsilon > C/\sqrt{n}$ for some constant C.*

4 Approximating Matching Statistics

We define the average size $E(G,\lambda)$ and the entropy $S(G,\lambda)$ of a matching as:

$$E(G,\lambda) = \sum_{M\in\mathbb{M}} |M|\, \pi_{G,\lambda}(M) \quad \text{and} \quad S(G,\lambda) = -\sum_{M\in\mathbb{M}} \pi_{G,\lambda}(M)\, \log \pi_{G,\lambda}(M).$$

The following algorithm estimates $E(G,\lambda)$, where C is a fixed constant.

```
Approx-Matching-Statistics($\lambda, \epsilon$)
1  $s \leftarrow \lceil C/\epsilon^2 \rceil$
2  $U \leftarrow$ random multi-subset of $V$ with $s$ elements
3  **return** $n - (n/2s) \cdot \sum_{u\in U}$(Approx-Marginal($\lambda, \epsilon/2, u$))
```

Theorem 7. Approx-Matching-Statistics(λ, ϵ) *is an ϵn-approximation algorithm for $E(G,\lambda)$ with query complexity $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$. In addition, any ϵn-approximation algorithm for $E(G,\lambda)$ needs $\Omega(1/\epsilon^2)$ queries.*

Proof. Let $A = \sum_{v\in V}$ Approx-Marginal($\lambda, \epsilon/2, v$). By Proposition 2, A is an $\epsilon n/2$-approximation of $\sum_{v\in V} p_{G,\lambda}(v)$. By Lemma 4, there exists some constant C such that approximating the marginal probability at $\lceil C/\epsilon^2 \rceil$ sampled nodes gives an $\epsilon n/2$-approximation of A with high probability. This implies an ϵn-approximation of $\sum_{v\in V} p_{G,\lambda}(v)$ with high probability. Since $E(G,\lambda) = n - \sum_{v\in V} p_{G,\lambda}(v)/2$, we thus get an ϵn-approximation of $E(G,\lambda)$ with high probability. The query complexity of this algorithm is $\lceil C/\epsilon^2 \rceil \cdot \overline{Q}(\epsilon/2, \Delta) = \tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$. The query lower bound is obtained similarly as Theorem 6. □

Corollary 8. *We have an ϵn-approximation algorithm for $S(G,\lambda)$ with query complexity $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$. In addition, any ϵn-approximation algorithm for $S(G,\lambda)$ needs $\Omega(1/\epsilon^2)$ queries.*

Proof. A simple calculation gives: $S(G,\lambda) = \log Z(G,\lambda) - \log\lambda \cdot E(G,\lambda)$. Let $\widehat{Z}$ be the output of Approx-Partition-Function($\lambda, \epsilon/2$) and $\widehat{E}$ be the output of Approx-Matching-Statistics($\lambda, \epsilon/(2\log\lambda)$). By Theorem 5 and Theorem 7, $\widehat{Z} - \log\lambda \cdot \widehat{E}$ is an ϵn-estimate of $S(G,\lambda)$ with high probability. Both $\widehat{Z}$ and $\widehat{E}$ are computed using $\tilde{O}\left((1/\epsilon)^{\tilde{O}(\sqrt{\Delta})}\right)$ queries. The query lower bound is obtained similarly as Theorem 6. □

5 Some Applications

So far, we did consider that the parameter λ is fixed. Letting λ grow with $\frac{1}{\epsilon}$ allows us to get new results for the permanent of a matrix. There is a FPRAS for the permanent of a matrix with non-negative entries [15]. When the matrix is the adjacency matrix of a *constant degree expander* graph, there is a PTAS to estimate the permanent within a multiplicative factor $(1+\epsilon)^n$ [8]. Using a key technical result of [8], we get a sublinear-time algorithm within the same multiplicative factor (see the full version of the paper).

Note that for a fixed graph G, if $\lambda \to \infty$ then the distribution $\pi_{G,\lambda}$ converges toward the uniform distribution on maximum matchings. Indeed, using a bound derived in [3], we can show that if λ grows exponentially with $\frac{1}{\epsilon}$ our technique allows to approximate the size of a maximum matching (see the full version of the paper). However our algorithm performs badly with respect to [32].

Acknowledgments. The authors acknowledge the support of the French *Agence Nationale de la Recherche (ANR)* under reference ANR-11-JS02-005-01 (GAP project).

References

1. Bayati, M., Gamarnik, D., Katz, D., Nair, C., Tetali, P.: Simple deterministic approximation algorithms for counting matchings. In: STOC, pp. 122–127. ACM (2007)
2. Blume, L.E.: The statistical mechanics of strategic interaction. Games Econom. Behav. 5(3), 387–424 (1993)
3. Bordenave, C., Lelarge, M., Salez, J.: Matchings on infinite graphs. Probability Theory and Related Fields 157(1-2), 183–208 (2013)
4. Canetti, R., Even, G., Goldreich, O.: Lower bounds for sampling algorithms for estimating the average. Information Processing Letters 53(1), 17–25 (1995)
5. Chazelle, B., Rubinfeld, R., Trevisan, L.: Approximating the minimum spanning tree weight in sublinear time. SIAM Journal on computing 34(6), 1370–1379 (2005)
6. Czumaj, A., Sohler, C.: Sublinear-time algorithms. Bulletin of the EATCS 89, 23–47 (2006)
7. Feige, U.: On sums of independent random variables with unbounded variance and estimating the average degree in a graph. SIAM Journal on Computing 35(4), 964–984 (2006)
8. Gamarnik, D., Katz, D.: A deterministic approximation algorithm for computing the permanent of a 0, 1 matrix. Journal of Computer and System Sciences 76(8), 879–883 (2010)
9. Gamarnik, D., Katz, D.: Correlation decay and deterministic FPTAS for counting list-colorings of a graph. In: SODA, pp. 1245–1254. SIAM (2007)
10. Godsil, C.D.: Matchings and walks in graphs. J. Graph Theory 5(3), 285–297 (1981)
11. Goldreich, O., Ron, D.: Approximating average parameters of graphs. Random Structures and Algorithms 32(4), 473–493 (2008)
12. Gonen, M., Ron, D., Shavitt, Y.: Counting stars and other small subgraphs in sublinear-time. SIAM Journal on Discrete Mathematics 25(3), 1365–1411 (2011)

13. Heilmann, O.J., Lieb, E.H.: Theory of monomer-dimer systems. Comm. Math. Phys. 25, 190–232 (1972)
14. Jerrum, M.: Counting, sampling and integrating: algorithms and complexity. Lectures in Mathematics ETH Zürich, Birkhäuser Verlag, Basel (2003)
15. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. Journal of the ACM 51(4), 671–697 (2004)
16. Jerrum, M., Valiant, L., Vazirani, V.: Random generation of combinatorial structures from a uniform distribution. Theoret. Comput. Sci. 43(2-3), 169–188 (1986)
17. Li, L., Lu, P., Yin, Y.: Approximate counting via correlation decay in spin systems. In: SODA, pp. 922–940. SIAM (2012)
18. Li, L., Lu, P., Yin, Y.: Correlation decay up to uniqueness in spin systems. In: SODA, pp. 67–84. SIAM (2013)
19. Lovász, L., Plummer, M.D.: Matching theory. AMS Chelsea Publishing, Providence (2009) corrected reprint of the 1986 original (MR0859549)
20. Mézard, M., Montanari, A.: Information, physics, and computation. Oxford Graduate Texts. Oxford University Press, Oxford (2009)
21. Nguyen, H., Onak, K.: Constant-time approximation algorithms via local improvements. In: FOCS, pp. 327–336. IEEE (2008)
22. Onak, K., Ron, D., Rosen, M., Rubinfeld, R.: A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In: SODA, pp. 1123–1131. SIAM (2012)
23. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theoret. Comput. Sci. 381(1-3), 183–196 (2007)
24. Rubinfeld, R.: Sublinear time algorithms. In: International Congress of Mathematicians, vol. III, pp. 1095–1110. Eur. Math. Soc., Zürich (2006)
25. Sinclair, A.: Algorithms for random generation and counting. In: Progress in Theoretical Computer Science, Birkhäuser Boston Inc., Boston (1993)
26. Sinclair, A., Srivastava, P., Thurley, M.: Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. In: SODA, pp. 941–953. SIAM (2012)
27. Sinclair, A., Srivastava, P.: Lee-yang theorems and the complexity of computing averages. In: STOC, pp. 625–634. ACM (2013)
28. Vadhan, S.P.: The complexity of counting in sparse, regular, and planar graphs. SIAM Journal on Computing 31(2), 398–427 (2002)
29. Weitz, D.: Counting independent sets up to the tree threshold. In: STOC, pp. 140–149. ACM (2006)
30. Welsh, D.J.A.: Complexity: knots, colourings and counting. London Mathematical Society Lecture Note Series, vol. 186. Cambridge University Press (1993)
31. Yao, A.: Probabilistic computations: Toward a unified measure of complexity. In: FOCS, pp. 222–227. IEEE (1977)
32. Yoshida, Y., Yamamoto, M., Ito, H.: An improved constant-time approximation algorithm for maximum independent sets and maximum matchings. In: STOC, pp. 225–234. ACM (2009)

The Complexity of Finding a Large Subgraph under Anonymity Constraints

Robert Bredereck[1,⋆], Sepp Hartung[1], André Nichterlein[1], and Gerhard J. Woeginger[2,⋆⋆]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin
[2] Department of Mathematics and Computer Science, TU Eindhoven

Abstract. We define and analyze an anonymization problem in undirected graphs, which is motivated by certain privacy issues in social networks. The goal is to remove a small number of vertices from the graph such that in the resulting subgraph every occurring vertex degree occurs many times.

We prove that the problem is NP-hard for trees, and also for a number of other highly structured graph classes. Furthermore we provide polynomial time algorithms for other graph classes (like threshold graphs), and thereby establish a sharp borderline between hard and easy cases of the problem. Finally we perform a parametrized analysis, and we concisely characterize combinations of natural parameters that allow FPT algorithms.

1 Introduction

With the tremendous usage of social networks, the protection of privacy when releasing underlying data sets has become an important and active field of research [15]. If a graph contains only few vertices with some distinguished feature, then this might allow the identification (and violation of privacy) of the underlying real world entities with that particular feature. Hence in order to ensure pretty good privacy and anonymity behavior, every vertex should share its features with many other vertices. In a landmark paper[1], Liu and Terzi [11] considered in their setting the vertex degrees as feature; see Wu et al. [15] for other features considered in the literature. Correspondingly, a graph is called k-anonymous if for each vertex there are at least $k-1$ other vertices of same degree. Therein, different values of k reflect different privacy demands and the natural computational task arises to perform few changes to a graph in order to make it k-anonymous.

Liu and Terzi [11] proposed an heuristic algorithm for the task of making a graph k-anonymous by adding edges. The same variant has been studied by

⋆ Supported by the DFG project PAWS (NI 369/10).

⋆⋆ Supported by DIAMANT (a mathematics cluster of the Netherlands Organization for Scientific Research NWO), and by the Alexander von Humboldt Foundation.

[1] According to Google Scholar, accessed on June 14, 2013, since its publication in 2008 the paper has been cited 256 times.

Hartung et al. [10] from a parameterized complexity perspective. In this paper, we complement these previous studies by investigating the vertex deletion variant which is defined as follows:

Degree Anonymity by Vertex Deletion (Anonym-V-Del)
Instance: An undirected graph $G = (V, E)$; positive integers k and s.
Question: Is there subset $S \subseteq V$ of size at most s such that deleting S in G results in a k-anonymous graph?

Considering vertex deletions instead of edge additions seems to be a promising approach on practical instances, especially on social networks. Therein, the degree distribution of the underlying graphs often follow a so-called power law distribution [1] implying that there are only few high degree vertices and most vertices are of moderate degree; this suggests that only few vertices have to be removed in order to get a k-anonymous graph. For instance, consider the DBLP co-author graph (generated in Feb. 2012) with ≈ 715 thousand vertices corresponding to authors and ≈ 2.5 million edges indicating whenever two authors have a common scientific paper: This graph has maximum degree 804 but only 208 vertices are of degree larger than 208, whereas the average degree is 7. Interestingly, a heuristic that simply removes vertices violating the k-anonymous property proves that one has to remove no more than 338 vertices to make it 5-anonymous and even to make it 10-anonymous requires at most 635 vertex deletions.

While there are many different privacy models, there is a lack of algorithms with provably good performance (as explicitly observed by [3]). In this work, we will show that already the simple and highly specialized privacy model of Anonym-V-Del is computationally hard from the parameterized as well as from the approximation point of view. A variety of hardness results holds even in very restricted graph classes, as for instance trees, cographs, and split graphs.

One reason for this hardness is shown in the following two examples illustrating that the number s of allowed removals and the degree k of anonymity are independent of each other, and that a small change in one of these parameters might lead to a large jump of the other parameter.

Example 1. Let G be a graph on $n \geq 5$ vertices that consists of two components: a clique of size $n - 2$ and a clique of size two. This 2-anonymous graph cannot be transformed into a 3-anonymous graph by deleting only one vertex, however, deleting two vertices makes it (n−2)-anonymous. Hence, by slightly increasing s from 1 to 2 the reachable anonymity-degree jumps from $k = 2$ to $k = n - 2$.

Example 2. Let $G = (V, E)$ be a graph with vertices $X = \{x_1, \ldots, x_\ell\}$ and $Y = \{y_1, \ldots, y_\ell\}$ with an edge between x_i and y_j if $i + j > \ell$. Clearly, x_i and y_i are of degree i implying that G is 2-anonymous. Since $N(x_i) \subseteq N(x_{i+1})$ for all i, deleting any subset of Y preserves the invariant $\deg(x_1) \leq \deg(x_2) \leq \ldots \leq \deg(x_\ell)$. As the previous argument is symmetric, one can observe that to make G 3-anonymous one has to remove $2/3$ of the "jumps" in the initial sequences $\deg(x_1) < \deg(x_2) < \ldots \deg(x_\ell)$ and $\deg(y_1) < \ldots < \deg(y_\ell)$. Since removing one vertex in X (Y) removes only one jump in the sequence of X (Y) and only one in Y (X), it follows that at least $2(\ell - 1) \cdot \frac{2}{3} \cdot \frac{1}{2} \approx \frac{2}{3}\ell = \frac{1}{3}|V|$ vertices have to be deleted in order to get a 3-anonymous graph. Summarizing, by requiring anonymity $k = 3$ instead of anonymity $k = 2$, the number of vertices needed to be removed jumps from zero to a constant fraction of vertices.

Related Work. Hartung et al. [10] studied the Anonym-E-Add problem as proposed by Liu and Terzi [11]. Given a graph and two positive integers k and s, Anonym-E-Add asks whether there exists a set of at most s edges whose addition makes the graph k-anonymous. The main result of Hartung et al. [10] is a polynomial problem kernel with respect to the parameter maximum degree Δ of the input graph. Furthermore, they showed that an heuristic algorithm proposed by Liu and Terzi [11] is optimal for Anonym-E-Add solutions larger than Δ^4. Chester et al. [4] investigated the computational complexity of Anonym-E-Add and variants with edge labels. They showed NP-hardness for the considered variants and a polynomial time algorithm for bipartite graphs.

Mathieson and Szeider [12] performed a parameterized complexity study for the problem of finding a minimum amount of graph editions in order to fulfill specified degree constraints. The graph editions considered are vertex deletion, edge insertion, edge deletions, and combinations thereof.

Our Results. Whereas every graph is trivially 1-anonymous, we will show that the combinatorial structure of 2-anonymous graphs is already rich and complicated: Anonym-V-Del for $k = 2$ is NP-hard, even for strongly restricted graph classes like trees, interval graphs, split graphs, trivially perfect graphs, and bipartite permutation graphs. All these hardness results are established by means of a general framework in Section 2. As a side-result, our framework implies the W[2]-hardness of various (natural) parameterized problem variants and the in-approximability of various (natural) optimization versions. Furthermore, we show that Anonym-V-Del is NP-hard even on graphs with maximum degree three; this result is in stark contrast with the fixed-parameter tractability of Anonym-E-Add with respect to the maximum degree Δ [10].

On the positive side, Section 3 presents (polynomial time) dynamic programming approaches for Anonym-V-Del on three graph classes: graphs of maximum degree two, P_3-free graphs, and threshold graphs. We frankly admit that these three graph classes carry an *extremely constraining* combinatorial structure: Anonym-V-Del is such a vicious problem that without these heavily constraining structures there remains no hope for polynomial time results. Figure 1 summarizes the considered graph classes and their containment relations.

Finally, we analyze the parametrized complexity of Anonym-V-Del in Section 4. Once again, Anonym-V-Del shows a difficult and challenging behavior: It is intractable with respect to each of the three (single) parameters s, k and Δ. Even worse, it is intractable with respect to the combined parameter (s, k). The only positive parametrized results come with the combined parameters (Δ, s) and (Δ, k). The latter result is based on bounding the number s of deleted vertices in terms of Δ and k.

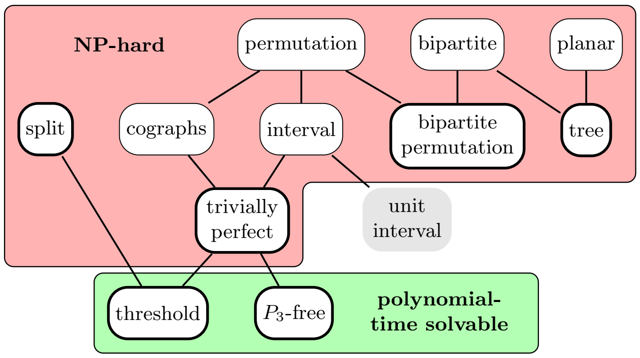


Fig. 1. The complexity landscape of Anonym-V-Del for various graph classes. The results for classes with thick frames are made in this work and they imply the results for classes with thin frames.

Preliminaries. All graphs in this paper are undirected, loopless, and simple (that is, without multiple edges). Throughout we use n to denote the number of vertices in the considered graph. The maximum vertex degree of a graph $G = (V, E)$ is denoted Δ_G. A vertex subset $S \subseteq V$ is called *k-deletion set* if $G[V \setminus S]$ is k-anonymous. For each vertex $v \in V$ we denote by $N_G(v)$ the set of neighbors of v and by $N_G[v] = N_G(v) \cup \{v\}$ the closed neighborhood. Correspondingly, for a vertex subset V' we set $N_G[V'] = \bigcup_{v \in V'} N_G[v]$ and $N_G(V') = N_G[V'] \setminus V'$. For $0 \leq a \leq \Delta$, the *block of degree a* is the set $D_G(a) \subseteq V$ of all vertices with degree a in G. Clearly, a graph is k-anonymous iff (if and only if) each block is either of size zero or at least k. We omit subscripts if the corresponding graph is clear from the context.

For the relevant notation of parameterized complexity and algorithmics we refer to the monographs of Downey and Fellows [7], Niedermeier [13]. Due to the space constraints some proofs are deferred to the appendix.

2 Computational Hardness

In this section we provide NP-hardness results for Anonym-V-Del on several restricted graph classes such as trees, split graphs, and trivially perfect graphs. As a warm up, we first prove that Anonym-V-Del is NP-hard on graphs with maximum degree three. This contrasts the known fixed-parameter tractability of Anonym-E-Add with respect to the parameter maximum degree [10].

Theorem 1. Anonym-V-Del *is NP-hard on graphs with degree at most three.*

Proof. We give a reduction from the Vertex Cover problem which is known to be NP-complete even in three-regular graphs [8, GT1]. Therein, given a three-regular graph together with an integer $h \in \mathbb{N}$ the task is to decide whether there is vertex set of size at most h such that each edge has at least one endpoint in it.

Given a VERTEX COVER instance $(G = (V, E), h)$, start by copying G to a new graph G'. Finally, add $h+1$ degree-zero vertices to G', set $s = h$, and $k = |V|+1$.

If G contains a vertex cover S of size h, then deleting S in G' clearly results in an edgeless graph with $|V| + 1 = k$ vertices, implying that (G', s, k) is a yes-instance of ANONYM-V-DEL. In the other direction, for any k-deletion set S, since $2k > n + h + 1$ and G' contains $s + 1$ degree-zero vertices, all vertices in $G' \setminus S$ have degree zero. Thus, $S \cap V$ is a vertex cover in G. □

NP-Hardness on Trees. Next we show that ANONYM-V-DEL is NP-hard even on trees. Extracting the basic ideas of this result, subsequently we provide a generic reduction to show NP-hardness on trivially perfect graphs, bipartite permutation graphs, and split graphs. Both reductions will reduce from the NP-hard SET COVER problem, which is defined as follows [8, SP5]: Given a universe $A = \{a_1, \ldots, a_\alpha\}$, a collection $\mathcal{B} = \{B_1, \ldots, B_\beta\}$ of sets over A, and $h \in \mathbb{N}$ the task is to decide whether there is an index set $I \subseteq \{1, \ldots, \beta\}$ with $|I| \leq h$, such that $\bigcup_{i \in I} B_i = A$?

Let $(A, \mathcal{B}, h)$ be an instance of SET COVER. We assume without loss of generality that for each element $a \in A$ there exists a set $B \in \mathcal{B}$ with $a \in B$. Furthermore, we assume without loss of generality that each set $B \in \mathcal{B}$ occurs at least $h + 2$ times in $\mathcal{B}$. To reduce the amount of indices in the construction given below we introduce the function $f\colon A \to \mathbb{N}$ that maps an element $a_i \in A$ to $f(a_i) = \alpha + (h + 1)i$.

The reduction for trees is as follows. Set $k = 2$ and $s = h$ such that (G, k, s) is an equivalent ANONYM-V-DEL-instance. Graph $G = (V, E)$ is constructed as follows: For each element $a_i \in A$ add an *element gadget* consisting of a star $K_{1,f(a_i)}$ with the center vertex $v(a_i)$. Denote with $V_A = \{v(a_1), \ldots, v(a_\alpha)\}$ the set of all these center vertices.

For each set $B_j \in \mathcal{B}$ add a *set gadget* which is a tree rooted in a vertex $v(B_j)$. The root has $|B_j|$ child vertices where each element $a_i \in B_j$ corresponds to exactly one of these children, denoted by $v(a_i, B_j)$. Additionally, we add to $v(a_i, B_j)$ exactly $f(a_i)$ degree-one neighbors. Hence, the set gadget is a tree of depth three rooted in $v(B_j)$. We denote with $V_{\mathcal{B}} = \{v(B_1), \ldots, v(B_\beta)\}$ the set of all root vertices. Observe that, as each set $B_j \in \mathcal{B}$ occurs at least $h + 2$ times, the set gadgets are $h + 2$-anonymous. Finally, to end up with one tree instead of a forest, repeatedly add edges between any degree-one-vertices of different connected components.

Observe that for each element $a_i \in A$ the only vertex of degree $f(a_i)$ is $v(a_i)$ and there are no other vertices violating the 2-anonymous property. The key point in the construction is that, in order to get a 2-anonymous graph, one has to delete vertices of $V_{\mathcal{B}}$: Let $a_i \in A$ be an element and $v(B_j)$ a root vertex such that $a_i \in B_j$. By construction the child vertex $v(a_i, B_j)$ of $v(B_j)$ corresponds to a_i and therefore has $f(a_i)$ child vertices. Thus, deleting $v(B_j)$ lowers the degree of $v(a_i, B_j)$ to $f(a_i)$ and, hence, $v(a_i)$ no longer violates the 2-anonymous property. Furthermore, as each set $B_j \in \mathcal{B}$ occurs at least $h + 2$ times, the vertices $V_{\mathcal{B}}$ are 2-anonymous. Hence, given a set cover one can construct a corresponding k-deletion set of the same size and, thus, if $(A, \mathcal{B}, h)$ is a yes-instance, then (G, k, s) is a yes-instance. The proof of the converse direction which implies the following theorem will be given later, after introducing the generic reduction.

Theorem 2. Anonym-V-Del *is NP-hard on trees even if* $k = 2$.

Generic Reduction. We now generalize the reduction given in the previous paragraph. More specifically, we will define properties such that a graph G fulfilling them together with $s = h$ and $k = 2$ forms a yes-instance of Anonym-V-Del iff the given Set Cover instance $(A, \mathcal{B}, h)$ is a yes-instance. Based on that, we then describe the construction of a several graphs contained in different graph classes and fulfilling the properties. Formally, we require the constructed graph $G = (V, E)$ to fulfill the following:

1. For each element $a_i \in A$ there is a corresponding vertex, denoted by $v(a_i)$, in G and the vertex set $V_A = \{v(a_1), \ldots, v(a_\alpha)\}$ is exactly the set of vertices not being 2-anonymous in G.
2. For each set $B_j \in \mathcal{B}$ there is a corresponding vertex $v(B_j)$ in G and for each element $a_i \in B_j$ the vertex $v(B_j)$ has a neighbor $v(a_i, B_j)$ with $\deg(v(a_i, B_j)) = \deg(v(a_i)) + 1$.
   Set $V_{\mathcal{B}} = \{v(B_1), \ldots, v(B_\beta)\}$ and $A_{B_j} = \{v(a_i, B_j) \mid a_i \in B_j\}$.
3. The vertex subsets V_A, $V_{\mathcal{B}}$, and $A_{B_1}, \ldots, A_{B_\beta}$ are pairwise disjoint. We set $A_{\mathcal{B}} = \bigcup_{B_j \in \mathcal{B}} A_{B_j}$.
4. For each $D \subseteq V_{\mathcal{B}}$, $|D| \leq h$, the set of vertices violating the 2-anonymous property in $G[V \setminus D]$ is a subset of V_A.
5. It holds: (a) $|N[v] \cap V_A| \leq 1$ for each vertex $v \in V$, (b) $N(A_{B_j}) \cap V_{\mathcal{B}} = \{v(B_j)\}$ for all $B_j \in \mathcal{B}$, and (c) $N(V_A) \cap (V_{\mathcal{B}} \cup A_{\mathcal{B}}) = \emptyset$.
6. For each vertex $v \in V$ there is a vertex $u \in V_{\mathcal{B}}$ such that $N(v) \cap A_{\mathcal{B}} \subseteq N(u)$.
7. Any two vertices $u \in V_A$ and $v \notin A_{\mathcal{B}}$ satisfy $|\deg(v) - \deg(u)| > s$.

It is not hard to verify that the graph constructed in the reduction in the previous paragraph has the above properties. Before proving the correctness of the generic reduction we show the following observation.

Observation 1. For each $D \subseteq V_{\mathcal{B}}$, $|D| \leq h$, the set $V_A \setminus \{v(a_i) \mid \exists v(B_j) \in D\colon a_i \in B_j\}$ is exactly the set of vertices not being 2-anonymous in $G[V \setminus D]$.

Lemma 1. *Let G be a graph satisfying Properties 1 to 7 for a given instance $(A, \mathcal{B}, h)$ of* Set Cover. *Then $(G, 2, h)$ is a yes-instance of* Anonym-V-Del *if and only if $(A, \mathcal{B}, h)$ is a yes-instance of* Set Cover.

Proof. If there is an index set I, $|I| \leq h$, such that $\bigcup_{j \in I} B_j = A$, then by Observation 1 the set $S = \{v(B_j) \mid j \in I\} \subseteq V_{\mathcal{B}}$, $|S| = |I|$, is a k-deletion set for G. It remains to prove the reverse direction.

Let S be a k-deletion set of size at most $s = h$ for $G = (V, E)$. We form a k-deletion set S' for G such that $S' \subseteq V_{\mathcal{B}}$ and $|S'| \leq |S|$. Consider each vertex $v \in S$: If $v \in V_{\mathcal{B}}$, then add v to S' (Case 1). If $v \in N[V_A]$, then by Property 5 there is only one a_i such that $v \in N[v(a_i)]$ and we add a vertex $v(B_j) \in V_{\mathcal{B}}$ with $a_i \in B_j$ to S' (Case 2). Finally, if $v \in N[A_{\mathcal{B}}]$, then by Property 6 there is a vertex $u \in V_{\mathcal{B}}$ with $N(v) \cap A_{\mathcal{B}} \subseteq N(u)$ and we add u to S' (Case 3).

We next prove that S' is a k-deletion set for G and thus by Observation 1 the index set corresponding to the vertices in S' is a solution of size $|S'|$ to the Set Cover instance.

Assume towards a contradiction that $G[V \setminus S']$ is not 2-anonymous. Denoting by $X \subseteq V \setminus S'$ the set of vertices not being 2-anonymous, it follows from Observation 1 that $X \subseteq V_A$. Moreover, by the construction of S' (see Case 2) and Observation 1 it follows that $N[X] \cap S = \emptyset$ and thus $\deg_{G[V \setminus S]}(u) = \deg_G(u)$ for all $u \in X$. Hence, for each $u \in X$ there is a vertex $w \in V$ such that $\deg_G(u) = \deg_{G[V \setminus S]}(w)$ and thus by Property 7 it follows that $w \in N[A_{\mathcal{B}}]$. This implies a contradiction to the construction of S' because from $w \in N[A_{\mathcal{B}}]$ it follows that S' contains w's neighbor in $V_{\mathcal{B}}$ (see Case 3) and thus $u \notin X$ by Observation 1. □

Using this generic reduction we now show NP-hardness on several graph classes which are defined as follows (see Brandstädt et al. [2]): *Trivially perfect graphs* are the (P_4, C_4)-free graphs, that is, they do not contain an induced path or cycle on four vertices. A graph G is a *bipartite permutation graph* if G is bipartite and does not contain an asteroidal triple (is AT-free). Three vertices of a graph form an asteroidal triple if every two of them are connected by a path avoiding the neighborhood of the third. A graph is a *split graph* if it can be partitioned into a clique and an independent set.

Theorem 3. Anonym-V-Del *is NP-hard on trivially perfect graphs, bipartite permutation graphs, and split graphs.*

Since Set Cover is W[2]-complete with respect to h [7] we have the following.

Corollary 1. Anonym-V-Del *is W[2]-hard with respect to parameter s, even if $k = 2$ and if the input graph is a tree, a bipartite permutation graph, a split graph, or a trivially perfect graph.*

Dom et al. [6] showed that Set Cover does not admit a polynomial kernel with respect to the combined parameter (α, h). Observe that in all above constructions except the one for split graphs we can bound s and Δ in a polynomial in α and h.

Corollary 2. Anonym-V-Del *on trees, bipartite permutation graphs or trivially perfect graphs does not admit a polynomial kernel with respect to the combined parameter (k, s, Δ).*

There are two natural optimization versions associated with Anonym-V-Del: in one version (called Max Anonym-V-Del) the goal is to maximize the anonymity k subject to the constraint that the number s of deleted vertices does not exceed a given bound; in the other version (called Min Anonym-V-Del) the goal is to minimize the number s of deleted vertices subject to the constraint that the anonymity does not go below a certain given bound. As Set Cover is NP-hard to approximate within a ratio $o(\log n)$ [14], the above reduction yields the following inapproximability result.

Corollary 3. *The optimization problem* Min Anonym-V-Del *on n-vertex graphs cannot be approximated within a factor of $o(\log n)$, unless $P = NP$.*

Since the above reduction gives NP-hardness for $k = 2$, we immediately get inapproximability within a factor of two for Max Anonym-V-Del.

Corollary 4. *The optimization problem* Max Anonym-V-Del *cannot be approximated*

- *within a factor of* $2 - \epsilon$*, unless* $P = NP$.
- *within a factor of* $2 - \epsilon$ *in* $f(s)n^{O(1)}$ *time for any computable* f*, unless FPT=W[2].*

3 Polynomially Solvable Cases

We complement our intractability results for Anonym-V-Del from Theorem 2 by showing that Anonym-V-Del is polynomial time solvable on graphs with maximum degree two, on graphs that are disjoint unions of cliques, and on threshold graphs.

3.1 Graphs with Maximum Degree Two

In contrast to graphs of maximum degree three (see Theorem 1), we observe that Anonym-V-Del is polynomial time solvable on graphs of maximum degree two. Note that a graph of maximum degree two is just a collection of paths and cycles. Given five integers d_0, d_1, d_2, x, y, it is easy to decide whether it is possible to remove x vertices from a path of length y (respectively, from a cycle of length y) such that there survive precisely d_0 vertices of degree zero, d_1 vertices of degree one, and d_2 vertices of degree two. A straight-forward dynamic programming approach based on this observation leads to the following.

Theorem 4. *On graphs of maximum degree two,* Anonym-V-Del *is polynomial time solvable.*

3.2 Disjoint Union of Cliques

Note that Anonym-V-Del is trivial on cliques: either the clique size is at least k, or otherwise one has to delete all the vertices. The following theorem shows that polynomial time solvability also carries over to the case where the graph is the disjoint union of several cliques. (Recall that a graph is the disjoint union of cliques if and only if it does not contain the 3-vertex path P_3 as an induced subgraph.)

Theorem 5. *On a* P_3*-free graph* G *with* n *vertices and maximum degree* Δ*,* Anonym-V-Del *can be solved in* $O(n^2\Delta)$ *time.*

3.3 A Polynomial Time Result for Threshold Graphs

We recall that a graph $G(V, E)$ is a *threshold graph* if there are positive real vertex weights $w(v)$ for $v \in V$, such that $\{v_1, v_2\} \in E$ if and only if $w(v_1) + w(v_2) \geq 1$; see Chvátal and Hammer [5] and Golumbic [9] for more information. Without loss of generality we will assume throughout that the vertex weights satisfy the following conditions:

- The vertex weights are pairwise distinct, and satisfy $0 < w(v) < 1$
- Any $v_1, v_2 \in V$ satisfy $w(v_1) + w(v_2) \neq 1$; in particular $w(v_1) \neq \frac{1}{2}$

Note that the closed neighborhoods in a threshold graph are totally ordered by inclusion: whenever $w(v_1) < w(v_2)$, then $N_G[v_1] \subseteq N_G[v_2]$ and consequently $\deg(v_1) \leq \deg(v_2)$.

Lemma 2. *Let $U \subseteq V$ be a subset of vertices with $|U| \geq 2$, let $w_{\min} = \min_{u \in U} w(u)$ and $w_{\max} = \max_{u \in U} w(u)$, and let $u_0, u_1 \in U$ be the vertices with $w(u_0) = w_{\min}$ and $w(u_1) = w_{\max}$. All vertices in U have identical degree, if and only if there is no vertex $v \in V \setminus \{u_0, u_1\}$ with $1 - w_{\max} < w(v) < 1 - w_{\min}$.*

Proof. Note that all vertices in U have identical degree, if and only if $N_G[u_0] = N_G[u_1]$. The latter condition in turn holds if and only if there is no vertex v in the graph (with $v \neq u_0$ and $v \neq u_1$) that is adjacent to u_1 but not to u_0, and this is equivalent to the stated condition $1 - w_{\max} < w(v) < 1 - w_{\min}$. □

Now consider some block U of constant degree in an optimal subgraph for Anonym-V-Del, and let $u_0, u_1 \in U$ and $w_{\min}$ and $w_{\max}$ be defined as in the lemma. The *territory* of this block is defined as the union of the two closed intervals $[w_{\min}, w_{\max}]$ and $[1 - w_{\max}, 1 - w_{\min}]$; note that these two intervals will overlap if $w_{\min} < \frac{1}{2} < w_{\max}$. The *canonical superset* $U^* \subseteq V$ consists of u_0 and u_1, together with all vertices $v \in V$ that satisfy $w_{\min} \leq w(v) \leq w_{\max}$ but not $1 - w_{\max} < w(v) < 1 - w_{\min}$. One message of Lemma 2 is that distinct blocks in an optimal subgraph must have disjoint territories. Another message of the lemma is that we may as well replace every block U by its canonical superset U^*: By adding these vertices, the degree in every block either remains the same or is uniformly increased by $|U^*| - |U|$. And if the territories of distinct blocks were disjoint before the replacement, then they will also be disjoint after the replacement. In other words, such a replacement does not violate k-anonymity but simplifies the combinatorial structure of the considered subgraph.

This suggests the following dynamic programming approach. For every real number r with $0 \leq r \leq \frac{1}{2}$, we consider the threshold graph G_r that is induced by the vertices $v \in V$ with $r \leq w(v) \leq 1 - r$; note that the only crucial values for r are the $O(n)$ values $w(v)$ and $1 - w(v)$ that fall between the bounds 0 and $\frac{1}{2}$. The goal is to compute for every graph G_r a largest k-anonymous subgraph. We start our computations with $r = \frac{1}{2}$ and work downwards towards $r = 0$.

The initialization step of the dynamic program handles subgraphs that consist of a single block whose territory contains the number $\frac{1}{2}$. Such a block will either be empty, or it is a canonical superset specified by two values $w_{\min}$ and $w_{\max}$. All in all, this only yields a polynomial number of cases to handle. In the main computation phase of the dynamic program, we consider a general graph G_r and check all possibilities for the outermost block, which is the block whose territory is farthest away from the center point $\frac{1}{2}$. Since this territory is the union of two intervals $[r, q]$ and $[1 - q, 1 - r]$, we may simply check all possibilities for the interval boundary q, and then combine the corresponding block with the (previously computed) largest k-anonymous subgraph for graph G_q. Since there is only a linear number $O(n)$ of candidate values for q, the largest k-anonymous subgraph of G_r can be found in linear time.

Theorem 6. *On threshold graphs with n vertices,* Anonym-V-Del *can be solved in $O(n^2)$ time.* □

4 Parametrized Results

Theorem 1, Theorem 2, and Corollary 1 show that there is no hope for fixed-parameter tractability neither for any of the individual parameters s, k or Δ nor for the combined parameter (s, k). In this subsection, we show that Anonym-V-Del becomes fixed-parameter tractable when considering the combined parameters (s, Δ) as well as (k, Δ). We start with a fixed-parameter algorithm for (s, Δ) and show that the minimum size of a solution is bounded by a function only depending on k and Δ.

Theorem 7. Anonym-V-Del *can be solved in $(s\Delta)^{O(s\Delta^2)}n^2 \log n$ time.*

Lemma 3. *For every yes-instance $(G = (V, E), k, s)$ of* Anonym-V-Del *with Δ denoting the maximum degree of G there is a subset $S \subseteq V$ with $|S| < 2^{\Delta}\Delta^3 2k$ such that $G[V \setminus S]$ is k-anonymous.*

By combining Theorem 7 and Lemma 3 we obtain fixed-parameter tractability with respect to the parameter (k, Δ): For an instance (G, k, s) of Anonym-V-Del apply the algorithm from Theorem 7 on $(G, k, \min\{s, 2^{\Delta}\Delta^3 2k\})$. The running time is bounded by $(2^{\Delta}\Delta^4 2k)^{O(2^{\Delta}\Delta^5 2k)}n^2 \log n$.

Corollary 5. Anonym-V-Del *is fixed-parameter tractable with respect to the combined parameter (k, Δ).*

5 Conclusion

In this paper, we have complemented the investigations of Hartung et al. [10] on the edge addition version of the degree anonymity problem to the vertex deletion version. To our surprise, there is a strong contrast in the complexity of the two problem versions: Whereas Anonym-E-Add admits a polynomial kernel with respect to the maximum degree [10], we proved NP-hardness of Anonym-V-Del on graphs with maximum degree three. Furthermore, bounding one of the input parameters s and k does not yield fpt-algorithms for Anonym-V-Del; however bounding the degree and bounding one of the input parameters s and k brings the problem into FPT.

Our results also provide a good view on the colorful complexity landscape of the vertex deletion version. We have shown that the problem is hard for most of the standard graph classes, and that one has to move on to highly structured classes like threshold graphs in order to get some polynomial time results. A number of questions remained open: What is the complexity of Anonym-V-Del on claw-free graphs? What is the complexity of Anonym-V-Del on unit interval graphs? Corollary 4 does not exclude the existence of a constant-factor approximation for Max Anonym-V-Del. Are there stronger inapproximability results? Can the bounds stated in Theorem 7 and Lemma 3 be improved?

References

1. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509 (1999)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: a Survey. SIAM Monographs on Discrete Mathematics and Applications, vol. 3. SIAM (1999)
3. Chester, S., Kapron, B., Srivastava, G., Venkatesh, S.: Complexity of social network anonymization. Social Network Analysis and Mining, 1–16 (2012a)
4. Chester, S., Kapron, B.M., Ramesh, G., Srivastava, G., Thomo, A., Venkatesh, S.: Why Waldo befriended the dummy? k-anonymization of social networks with pseudo-nodes. In: Social Network Analysis and Mining, pp. 1–19 (2012b) ISSN 1869-5450
5. Chvátal, V., Hammer, P.L.: Aggregation of inequalities in integer programming. Annals of Discrete Mathematics 1, 145–162 (1977)
6. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through colors and IDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman (1979)
9. Golumbic, M.C.: Algorithmic graph theory and perfect graphs, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier B.V. (2004), 1st edn. Academic Press (1980)
10. Hartung, S., Nichterlein, A., Niedermeier, R., Suchý, O.: A refined complexity analysis of degree anonymization in graphs. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 594–606. Springer, Heidelberg (2013)
11. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: Proc. ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 93–106. ACM (2008)
12. Mathieson, L., Szeider, S.: Editing graphs to satisfy degree constraints: A parameterized approach. J. Comput. Syst. Sci. 78(1), 179–191 (2012)
13. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
14. Vazirani, V.V.: Approximation Algorithms. Springer (2001)
15. Wu, X., Ying, X., Liu, K., Chen, L.: A survey of privacy-preservation of graphs and social networks. In: Managing and Mining Graph Data, pp. 421–453. Springer (2010)

On the Number of Edges of Fan-Crossing Free Graphs*

Otfried Cheong[1], Sariel Har-Peled[2], Heuna Kim[3], and Hyo-Sil Kim[4]

[1] Department of Computer Science, KAIST, Daejeon, Korea
otfried@kaist.edu
[2] Department of Computer Science, University of Illinois, Urbana, USA
sariel@uiuc.edu
[3] Freie Universität Berlin, Berlin, Germany
heunak@mi.fu-berlin.de
[4] Department of Computer Science and Engineering, POSTECH, Pohang, Korea
hyosil.kim@gmail.com

Abstract. A graph drawn in the plane with n vertices is *fan-crossing free* if there is no triple of edges e, f and g, such that e and f have a common endpoint and g crosses both e and f. We prove a tight bound of $4n - 9$ on the maximum number of edges of such a graph for a straight-edge drawing. The bound is $4n - 8$ if the edges are Jordan curves. We also discuss generalizations to monotone graph properties.

Keywords: graph theory, graph drawing, planar graph, extremal graph.

1 Introduction

A *topological graph* G is a graph drawn in the plane: vertices are points in the plane, and the edges of the graph are drawn as Jordan curves connecting the vertices. Edges are not allowed to pass through vertices other than their endpoints. We will assume the topological graph to be *simple*, that is, any pair of its edges have at most one point in common (so edges with a common endpoint do not cross, and edges cross at most once). Figure 1 (a–b) shows configurations that are not allowed.

If there are no crossings between edges, then the graph is planar, and Euler's formula implies that it has at most $3n - 6$ edges, where n is the number of vertices. What can be said if we relax this restriction—that is, we permit some edge crossings?

For instance, a topological graph is called *k-planar* if each edge is crossed at most k times. Pach and Tóth [13] proved that a k-planar graph on n vertices has at most $(k + 3)(n - 2)$ edges for $0 \leqslant k \leqslant 4$, and at most $4.108\sqrt{k}n$ edges for general k. The special case of 1-planar graphs has recently received some attention, especially in the graph drawing community. Pach and Tóth's bound

* This research was supported in part by NRF grant 2011-0016434 and in part by NRF grant 2011-0030044 (SRC-GAIA), both funded by the government of Korea.

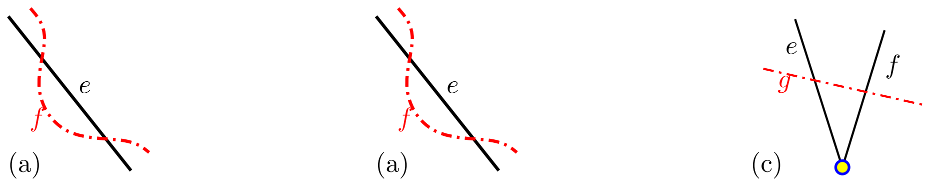


Fig. 1. (a) and (b) shows illegal embeddings of edges of a graph. (c) is a fan crossing.

is $4n-8$, and this is tight: starting with a planar graph H where every face is a quadrilateral, and adding both diagonals results in a 1-planar graph with $4n-8$ edges. However, Didimo [4] showed that *straight-line* 1-planar graphs have at most $4n-9$ edges, showing that Fáry's theorem does not generalize to 1-planar graphs. Hong et al. [8] characterize the 1-planar graphs that can be drawn as straight-line 1-planar graphs. Didimo's bound is tight, as he constructed an infinite family of straight-line 1-planar graphs with $4n-9$ edges. Korzhik and Mohar [9] showed that testing if a given graph is 1-planar is NP-hard.

A topological graph is called *k-quasi planar* if it does not contain k pairwise crossing edges. It is conjectured that for any fixed k the number of edges of a k-quasi planar graph is linear in the number of vertices n. Agarwal et al. [2] proved this for straight-line 3-planar graphs, Pach et al. [11] for general 3-planar graphs, Ackerman [1] for 4-planar graphs, and Fox et al. [7] prove a bound of the form $O(n \log^{1+o(1)} n)$ for k-planar graphs.

A different restriction on crossings arises in graph drawing: Humans have difficulty reading graph drawings where edges cross at acute angles, but graph drawings where edges cross at right angles are nearly as readable as planar ones. A *right-angle crossing graph* (RAC graph) is a topological graph with straight edges where edges that cross must do so at right angle. Didimo et al. [5] showed that an RAC graph on n vertices has at most $4n-10$ edges. Testing whether a given graph is an RAC graph is NP-hard [3]. Eades and Liotta [6] showed that an extremal RAC graph, that is, an RAC graph with n vertices and $4n-10$ edges, is 1-planar, and is the union of two maximal planar graphs sharing the same vertex set.

A *radial (p,q)-grid* in a graph G is a set of $p+q$ edges such that the first p edges are all incident to a common vertex, and each of the first p edges crosses each of the remaining q edges. Pach et al. [10] proved that a graph without a radial (p,q)-grid, for $p,q \geqslant 1$, has at most $8 \cdot 24^q pn$ edges. We call a radial $(2,1)$-grid a *fan crossing.* In other words, a fan crossing is formed by an edge g crossing two edges e and f that are incident to a common vertex, see Figure 1 (c). A topological graph is *fan-crossing free* if it does not contain a fan crossing.

By Pach et al.'s result, a fan-crossing free graph on n vertices has at most $384n$ edges. We improve this bound by proving the following theorem.

Theorem 1. *A fan-crossing free graph on $n \geqslant 3$ vertices has at most $4n-8$ edges. If the graph has straight edges, it has at most $4n-9$ edges. Both bounds are tight for $n \geqslant 10$.*

A 1-planar graph is fan-crossing free, so Theorem 1 generalizes both Pach and Tóth's and Didimo's bound. We also extend their lower bounds by giving tight constructions for every value of n.

In an RAC graph all edges crossed by a given edge g are orthogonal to it and therefore parallel to each other, implying that an RAC graph is fan-crossing free. Our theorem, therefore, "nearly" implies Didimo et al.'s bound: a fan-crossing free graph has at most one edge more than an RAC graph.

We can completely characterize extremal fan-crossing free graphs, that is, fan-crossing free graphs on n vertices with $4n-8$ edges: Any such graph consists of a planar graph H where each face is a quadrilateral, together with both diagonals for each face. This implies the same properties obtained by Eades and Liotta for extremal RAC graphs: An extremal fan-crossing free graph is 1-planar, and is the union of two maximal planar graphs.

Most of the graph families discussed above have a common pattern: the subgraphs obtained by taking the edges crossed by a given edge e may not contain some forbidden subgraph. We can formalize this notion as follows: For a topological graph G and an edge e of G, let G_e denote the subgraph of G containing exactly those edges that cross e.

A graph property $\mathcal{P}$ is called *monotone* if it is preserved under edge-deletions. In other words, if G has $\mathcal{P}$ and G' is obtained from G by deleting edges, then G' must have $\mathcal{P}$. Given a monotone graph property $\mathcal{P}$, we define a *derived graph property* $\mathcal{P}^*$ as follows: A topological graph G has $\mathcal{P}^*$ if for every edge e of G the subgraph G_e has $\mathcal{P}$. Some examples are:

- If $\mathcal{P}$ is the property that a graph does not contain a path of length two, then $\mathcal{P}^*$ is the property of being fan-crossing free;
- if $\mathcal{P}$ is the property of having at most k edges, then $\mathcal{P}^*$ is k-planarity;
- if $\mathcal{P}$ is planarity, then $\mathcal{P}^*$ is 3-quasi-planarity.

We can consider $\mathcal{P}^*$ for other interesting properties $\mathcal{P}$, such as not containing a path of length k, or not containing a $K_{2,2}$.

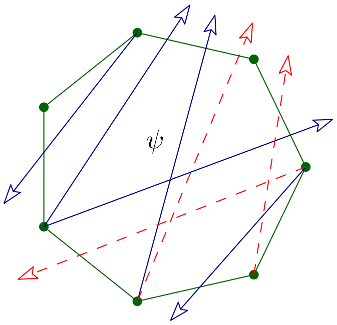
We prove the following very general theorem:

Theorem 2. *Let $\mathcal{P}$ be a monotone graph property such that any graph on n vertices that has $\mathcal{P}$ has at most $O(n^{1+\alpha})$ edges, for a constant $0 \leqslant \alpha \leqslant 1$. Let G be a graph on n vertices that has $\mathcal{P}^*$. If $\alpha > 0$, then G has $O(n^{1+\alpha})$ edges. If $\alpha = 0$, then G has $O(n \log^2 n)$ edges.*

This immediately covers many interesting cases. For instance, a graph where no edge crosses a path of length k, for a constant k, has at most $O(n \log^2 n)$ edges. Graphs where no edge crosses a $K_{2,2}$ have at most $\Theta(n^{3/2})$ edges (and this is tight, as there are graphs with $\Theta(n^{3/2})$ edges that do not contain a $K_{2,2}$, implying that no edge can cross a $K_{2,2}$).

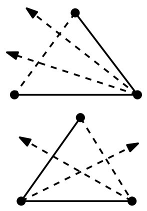
2 A Combinatorial Puzzle

At the core of our bound lies a combinatorial question that we can express as follows: An *m-star* is a regular m-gon ψ with a set of *arrows*. An arrow is a ray starting at a vertex of ψ, pointing into the interior of ψ, and exiting through an edge of ψ.



We require the set of edges and arrows to be *fan-crossing free*—that is, no edge or arrow intersects two arrows or an edge and an arrow incident to the same vertex. The side figure shows a 7-star. The dashed arrows are impossible—each of them forms a fan crossing with the solid edges and arrows.

The question is: *How many arrows can an m-star possess?*



Observation 1. *A 3-star has at most one arrow.*

Proof. An arrow from a vertex v has to exit the triangle ψ through the opposing edge, so no vertex has two arrows. But two arrows from different vertices will also form a fan crossing, see the side figure. □

It is not difficult to see that a 4-star possesses at most 2 arrows. The reader may enjoy constructing m-stars with $2m - 6$ arrows, for $m \geqslant 4$. We conjecture that this bound is tight. In the following, we will only prove a weaker bound that is sufficient to obtain tight results for fan-crossing free graphs.

While we have posed the question in a geometric setting, it is important to realize that it is a *purely combinatorial* question. We can represent the m-star by writing its sequence of vertices and indicating when an arrow exits ψ. Whether or not three edges/arrows form a fan crossing can be determined from the ordering of their endpoints along the boundary of ψ alone.

Let $C = v_1, \ldots, v_m$ be the sequence of vertices of ψ in counter-clockwise order, such that the ith boundary edge of ψ is $e_i = v_i v_{i+1}$ (all indices are modulo m). Consider an arrow e starting at v_i. It exits ψ through some edge e_j, splitting ψ into two chains $v_{i+1} \ldots v_j$ and $v_{j+1} \ldots v_{i-1}$. The *length* of e is the number of vertices on the shorter chain.

We will call an arrow *short* if it has length one. A *long arrow* is an arrow of length larger than one.

Lemma 1. *For $m \geqslant 4$, an m-star ψ has at most $2m - 8$ long arrows.*

Proof. The proof is by induction over m.

Any arrow in a 4-star partitions the boundary into chains of length one and length two, and so there are no long arrows, proving the claim for $m = 4$.

We suppose now that $m > 4$ and that the claim holds for all $4 \leqslant m' < m$. We *delete all short arrows*, and let L denote the remaining set of arrows, all of which are now long arrows. Let e be an arrow of *shortest length* ℓ in ψ. Without loss of generality, we assume that e starts in v_1 and exits through edge $e_{\ell+1} = v_{\ell+1} v_{\ell+2}$. Then, the following properties hold (see Figure 2):

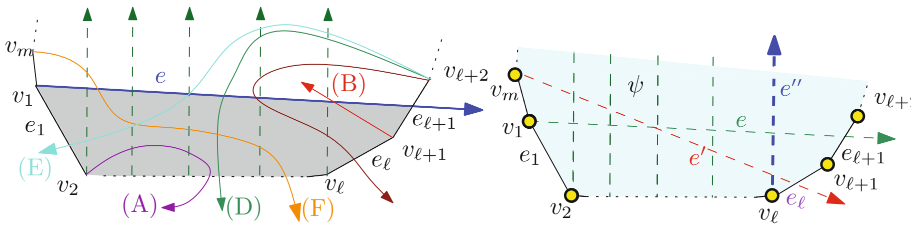


Fig. 2. Left: properties (A) to (F), right: property (G) for the proof of Lemma 1.

(A) Every arrow starting in $v_2, \dots, v_{\ell+1}$ must cross e, as otherwise it would be shorter than e.
(B) There is no arrow that starts in $v_{\ell+1}$. By (A), such an arrow must cross e, and so it forms a fan crossing with e and $e_{\ell+1}$.
(C) At most one arrow starts in v_i, for $i = 2, \dots, \ell$. Indeed, two arrows starting in v_i, for $i = 2, \dots, \ell$, must cross e by (A), and so they form a fan crossing with e.
(D) No arrow starting in $v_{\ell+2}$ exits through $e_2, \dots, e_\ell$, as then it would be shorter than e.
(E) An arrow starting in $v_{\ell+2}$ and exiting through e_1 cannot exist either, as it forms a fan crossing with e and e_1.
(F) No arrow starting in v_m crosses $e_1, \dots, e_{\ell-1}$, as then it would be shorter than e.
(G) The following two arrows cannot both exist: An arrow e' starting in v_m and exiting through e_ℓ, and an arrow e'' starting in v_ℓ. Indeed, if both e' and e'' are present, then either e'' exits through e_m and forms a fan crossing with e and e_m, or e'' intersects e' and so e', e'', and e_ℓ form a fan crossing (see the right side of Figure 2).

We now create an $(m - \ell + 1)$-star φ by removing the vertices $v_2 \dots v_\ell$ with all their incident arrows from ψ, such that v_1 and $v_{\ell+1}$ are consecutive on the boundary of φ. An arrow that exits ψ through one of the edges $e_1 \dots e_\ell$ exits φ through the new edge $g = v_1 v_{\ell+1}$.

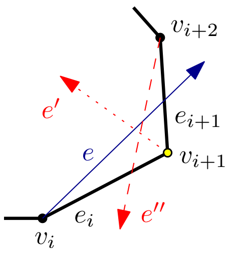
Let $L' \subset L$ be the set of arrows of φ, that is, the arrows of ψ that do not start from $v_2 \dots v_\ell$. Among the arrows in L', there are one or two short arrows: the arrow e, and the arrow e' starting in v_m and exiting through e_ℓ in ψ (and therefore through g in φ) if it exists. We set $q = 1$ if e' exists, and else $q = 0$.

We delete from φ those one or two short arrows, and claim that there is now no fan crossing in φ. Indeed, a fan crossing would have to involve the new edge $g = v_1 v_{\ell+1}$. But any arrow that crosses g must also cross e, and there is no arrow starting in $v_{\ell+1}$ by (B).

Since $\ell \geqslant 2$, we have $m - \ell + 1 < m$, and so by the inductive assumption φ has at most $2(m - \ell + 1) - 8 = 2m - 2\ell - 6$ long arrows. Since there are $1 + q$ short arrows in L', we have $|L'| \leqslant 2m - 2\ell - 5 + q$. By (C) and (G), we have

$|L| - |L'| \leqslant \ell - 1 - q$. It follows that

$$|L| \leqslant |L'| + \ell - 1 - q \leqslant 2m - \ell - 6 \leqslant 2m - 8. \quad \square$$



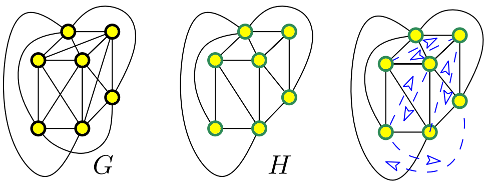
It remains to count the short arrows. Let e be a short arrow, say starting in v_i and exiting through e_{i+1}. Let us call v_{i+1} the *witness* of e. We observe that no arrow e' can start in this witness—e' would form a fan crossing with e and e_{i+1}. The vertex v_{i+1} can serve as the witness of only one short arrow: The only other possible short arrow e'' with witness v_{i+1} starts in v_{i+2} and exits through e_i. However, e, e'', and e_i form a fan crossing.

We can now bound the number of arrows of an m-star.

Lemma 2. *For $m \geqslant 3$, an m-star ψ has at most $3m - 8$ arrows. The bound is attained only for $m = 3$.*

Proof. By Observation 1, the claim is true for $m = 3$. We consider $m > 3$. By Lemma 1, there are at most $2m - 8$ long arrows. Each short arrow has a unique witness. If all vertices are witnesses then there is no arrow, and so we can assume that at most $m - 1$ vertices serve as witnesses, and we have at most $m - 1$ short arrows, for a total of $3m - 9$ arrows. $\square$

3 The Upper Bound

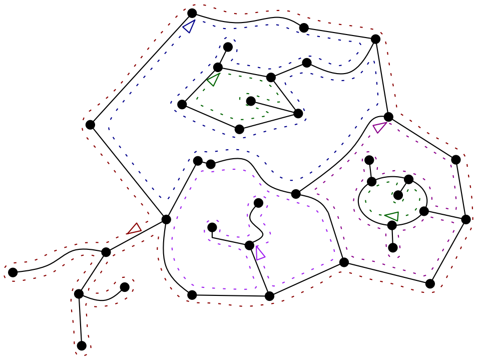


Let $G = (V, E)$ be a fan-crossing free graph. We fix an arbitrary maximal planar subgraph $H = (V, E')$ of G. Let $K = E \setminus E'$ be the set of edges of G that is not in H. Since H is maximal, every edge in K must cross at least one edge of H. We will replace each edge of K by two *arrows*.

Let $e \in K$ be an edge connecting vertices v and u. The initial segment of e must lie inside a face ψ of H incident to v, the final segment must lie inside a face φ of H incident to u. It is possible that $\psi = \varphi$, but in that case the edge e does not entirely lie in the face. We replace e by two arrows: one arrow starting in v and passing through ψ until it exits ψ through some edge; another arrow starting in u and passing through φ until it exits φ through some edge.

In this manner, we replace the set of edges K by a set of $2|K|$ arrows. The result is a planar graph whose faces have been adorned with arrows. The collection of edges and arrows is fan-crossing free.

Every edge of H is incident to two faces of H, which can happen to be identical. If we distinguish the sides of an edge, the boundary of each face ψ of H consists of simple chains of edges. If ψ is bounded, one chain bounds ψ from the outside, while all other chains bound holes inside ψ; if ψ is unbounded, then all chains bound holes in ψ.



If the graph H is connected, then the boundary of each face consists of a single chain. Let ψ be such a face whose boundary chain consists of m edges (where edges that bound ψ on both sides are counted twice). Then ψ has at most $3m - 8$ arrows. This follows immediately from Lemma 2: Recall that m-stars can be defined purely combinatorially. Whether three edges form a fan crossing can be decided solely by the ordering of their endpoints along the boundary chain. The boundary of a simply connected face is a single closed chain, and so Lemma 2 applies to this setting, see the side figure.

Unfortunately, we cannot guarantee that H is connected. The following lemma bounds the number of arrows of a face ψ in terms of its complexity and its number of boundary chains. The complexity of a face is the total number of edges of all its boundary chains, where edges that are incident to the face on both sides are counted twice.

Lemma 3. *A face of H of complexity m bounded by p boundary chains possesses at most $3m+8p-16$ arrows. The bound can be attained only when $m = 3$ and $p = 1$.*

We will prove the lemma below, but let us first observe how it implies the upper bound on the number of edges of fan-crossing free graphs.

Lemma 4. *A fan-crossing free graph G on n vertices has at most $4n - 8$ edges.*

Proof. Let m be the number of edges, let r be the number of faces, and let p be the number of connected components of H. Let $\mathcal{F}$ be the set of faces of H. For a face $\psi \in \mathcal{F}$, let $m(\psi)$ denote the complexity of ψ, let $p(\psi)$ denote the number of boundary chains of ψ, and let $a(\psi)$ denote the number of arrows of ψ.

We have $\sum_{\psi\in\mathcal{F}} m(\psi) = 2m$ and $\sum_{\psi\in\mathcal{F}}(p(\psi) - 1) = p - 1$ (each component is counted in its unbounded face, except that we miss one hole in the global unbounded face).

The graph G has $z = m + |K|$ edges. Using Lemma 3 we have

$$\begin{aligned}
2z = 2m + 2|K| &= \sum_{\psi\in\mathcal{F}} m(\psi) + \sum_{\psi\in\mathcal{F}} a(\psi) \\
&\leqslant \sum_{\psi\in\mathcal{F}} \big(4m(\psi) + 8p(\psi) - 16\big) \\
&= 4\sum_{\psi\in\mathcal{F}} m(\psi) + 8\sum_{\psi\in\mathcal{F}} (p(\psi) - 1) - 8r \\
&= 8m + 8p - 8 - 8r.
\end{aligned}$$

By Euler's formula, we have $n - m + r = 1 + p$, so $m - r = n - 1 - p$, and we have

$$2z \leqslant 8(m - r) + 8p - 8 = 8n - 8 - 8p + 8p - 8 = 8n - 16. \quad \square$$

It remains to fill in the missing proof.

Proof (of Lemma 3). Let ψ be a face of H, and let $m = m(\psi)$ and $p = p(\psi)$ be its complexity and its number of boundary components. A boundary component is a chain of edges, and could possibly degenerate to a single isolated vertex.

We say that two boundary chains ξ and ζ are *related* if an arrow starting in a vertex of ξ ends in an edge of ζ, or vice versa. Consider the undirected graph whose nodes are the boundary chains of ψ and whose arcs connect boundary chains that are related. If this graph has more than one connected component, we can bound the number of arrows separately for each component, and so in the following we can assume that all boundary chains are (directly or indirectly) related.

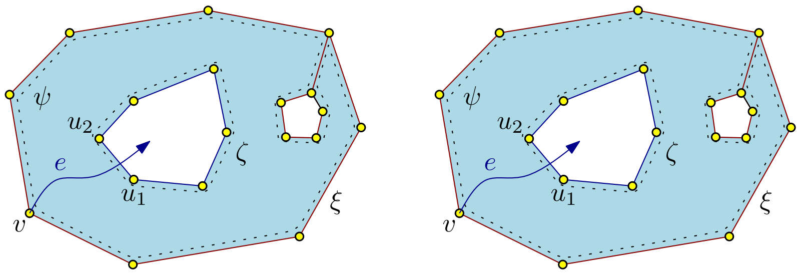


Fig. 3. Building a bridge between ξ and ζ

Consider two related boundary chains ξ and ζ. By assumption there must be an arrow e, starting at a vertex $v \in \xi$, and ending in an edge u_1u_2 of ζ. We create a new vertex z on ζ at the intersection point of e and u_1u_2, split the boundary edge u_1u_2 into two edges u_1z and zu_2, and insert the two new boundary edges vz and zv, see Figure 3. This operation has increased the complexity of ψ by three. Note that some arrows of ψ might be crossing the new boundary edges—these arrows will now be shortened, and end on the new boundary edge.

The two boundary chains ξ and ζ have now merged into a single boundary chain. In effect, we have turned an arrow into a "bridge" connecting two boundary chains. No fan crossing is created, since all edges and arrows already existed. We do create a new vertex z, but no arrow starts in z, and so this vertex cannot cause a fan crossing.

We insert $p - 1$ bridges in total and connect all p boundary chains. In this manner, we end up with a face φ whose boundary is a single chain consisting of $m' = m + 3(p - 1)$ edges.

If $m' = 3$, then φ has at most one arrow, by Observation 1. This case happens only for $m = 3$ and $p = 1$, and is the only case where the bound is tight.

If $m' > 3$, then we can apply Lemma 1 to argue that φ has at most $2m' - 8$ long arrows. To count the short arrows, we observe that the vertex z created in the bridge-building process cannot be the witness of a short arrow: such a short arrow would imply a fan crossing in the original face ψ. It is also not the starting point of any arrow.

It follows that building a bridge increases the number of possible witnesses by only one (the vertex v now appears twice on the boundary chain). There are thus at most $m + p - 1$ possible witnesses in φ. However, if all of these vertices are witnesses, then there is no arrow at all, and so there are at most $m + p - 2$ short arrows.

Finally, we converted $p - 1$ arrows of ψ into bridges to create φ. The total number of arrows of ψ is therefore at most

$$\begin{aligned} 2m' - 8 + (m + p - 2) + (p - 1) &= 2(m + 3p - 3) - 8 + (m + p - 2) + (p - 1) \\ &= 3m + 8p - 17. \quad \square \end{aligned}$$

For reasons of space we have to omit the characterization of extremal fan-crossing free graphs and the lower bounds in this extended abstract. We state the following lemma:

Lemma 5. *A fan-crossing free graph G with $4n - 8$ edges contains a planar graph Q on its vertex set, where each face of Q is a quadrilateral. G is obtained from Q by adding both diagonals for each face of Q.*

Lemma 5 implies the bound for straight-line graphs:

Lemma 6. *A fan-crossing free graph drawn with straight edges has at most $4n - 9$ edges. This bound is tight for $n \geqslant 6$.*

Finally, we can give tight lower-bound constructions for every value of n.

Lemma 7. *Extremal fan-crossing free graphs with $4n - 8$ edges exist for $n = 8$ and all $n \geqslant 10$. For $n \in \{7, 9\}$, extremal fan-crossing free graphs have $4n - 9$ edges.*

4 The General Bound

We now prove Theorem 2. The proof makes use of the following lemma by Pach et al. [12]:

Lemma 8 ([12, Theorem 2.1]). *Let G be a graph with n vertices of degree $d_1, \dots, d_n$ and crossing number χ. Then there is a subset E of b edges of G such that removing E from G creates components of size at most $2n/3$, and*

$$b^2 \leqslant (1.58)^2 \big(16\chi + \sum_{i=1}^{n} d_i^2\big).$$

Proof (of Theorem 2). Let G be a graph on n vertices with m edges having property $\mathcal{P}^*$. Since each edge e crosses a graph that has property $\mathcal{P}$, the crossing number of G is at most $\chi \leqslant O(mn^{1+\alpha})$. The degree of any vertex is bounded by $n-1$, and so we have $d_i^2 \leqslant n \cdot d_i$. It follows using Lemma 8 that there exists a set E of b edges in G such that

$$b^2 \leqslant O(\chi + \sum_{i=1}^{n} d_i^2) \leqslant O(mn^{1+\alpha} + n\sum_{i=1}^{n} d_i) \leqslant O(mn^{1+\alpha} + mn) \leqslant O(mn^{1+\alpha}),$$

and removing E from G results in components of size at most $2n/3$.

We recursively subdivide G. Level 0 of the subdivision is G itself. We obtain level $i+1$ from level i by decomposing each component of level i using Lemma 8.

Consider a level i. It consists of k components $G_1, \ldots, G_k$. Component G_j has n_j vertices and m_j edges, where $n_j \leqslant (\frac{2}{3})^i n$. The total number of edges at level i is $r = \sum_{j=1}^{k} m_j$.

Using the Cauchy-Schwarz inequality for the vectors $\sqrt{m_j}$, $\sqrt{n_j}$, we have

$$\sum_{j=1}^{k} \sqrt{m_j n_j} \leqslant \sqrt{\sum_{j=1}^{k} m_j}\sqrt{\sum_{j=1}^{k} n_j} = \sqrt{rn} \leqslant \sqrt{mn}.$$

We first consider the case $\alpha > 0$. The number of edges needed to subdivide G_j is $O(\sqrt{m_j n_j^{1+\alpha}})$. We bound this using $n_j \leqslant (\frac{2}{3})^i n$ as $O(\sqrt{m_j n_j}((\frac{2}{3})^i n)^{\alpha/2})$, and we obtain that the total number of edges removed between levels i and $i+1$ is bounded by $O(\sqrt{mn}((\frac{2}{3})^i n)^{\alpha/2})$. Since $(\frac{2}{3})^{\alpha/2} < 1$, summing over all levels results in a geometric series, and so the total number of edges removed is $O(\sqrt{mn^{1+\alpha}})$. But this implies that the total number of edges in the graph is bounded as

$$m \leqslant O(\sqrt{mn^{1+\alpha}}),$$

and squaring both sides and dividing by m results in

$$m \leqslant O(n^{1+\alpha}).$$

Next, consider the case $\alpha = 0$. The number of edges removed between levels i and $i+1$ is bounded by $O(\sqrt{mn})$. Adding over all $O(\log n)$ levels shows that

$$m \leqslant O(\sqrt{mn} \log n).$$

Again, squaring and dividing by m leads to

$$m \leqslant O(n \log^2 n). \quad \square$$

5 Conclusions

A natural next question to ask is if our techniques can be used for graphs that do not contain a radial (p, q)-grid, for other values of p and q, and if we can find tighter bounds than Pach et al. [10]. It is not difficult to generalize our argument to the case of graphs without radial $(p, 1)$-grids, but we do not seem to be able to obtain tight bounds yet.

In Theorem 2, we have given a rather general bound on the number of edges of graphs that exclude certain crossing patterns. The theorem shows that for graph properties $\mathcal{P}$ that imply that the number of edges grows as $\Theta(n^{1+\alpha})$, for $\alpha > 0$, the size of the entire graph is bounded by the same function. For the interesting case $\alpha = 0$, which arises for instance for fan-crossing free graphs, our bound includes an extra $\log^2 n$-term. Is this term an artifact of our proof technique, or are the examples of graph properties where $\mathcal{P}$ implies a linear number of edges, but graphs with $\mathcal{P}^*$ and a superlinear number of edges exist?

Acknowledgments. For helpful discussions, we thank Antoine Vigneron, Yoshio Okamoto, and Shin-ichi Tanigawa, as well as the other participants of the Korean Workshop on Computational Geometry 2011 in Otaru, Japan.

References

1. Ackerman, E.: On the maximum number of edges in topological graphs with no four pairwise crossing edges. Discrete & Computational Geometry 41(3), 365–375 (2009)
2. Agarwal, P.K., Aronov, B., Pach, J., Pollack, R., Sharir, M.: Quasi-planar graphs have a linear number of edges. Combinatorica 17, 1–9 (1997)
3. Argyriou, E.N., Bekos, M.A., Symvonis, A.: The straight-line RAC drawing problem is NP-hard. J. Graph Algorithms Appl. 16(2), 569–597 (2012)
4. Didimo, W.: Density of straight-line 1-planar graph drawings. Inf. Process. Lett. 113(7), 236–240 (2013)
5. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theo. Comp. Sci. 412(39), 5156–5166 (2011)
6. Eades, P., Liotta, G.: Right angle crossing graphs and 1-planarity. Discrete Applied Mathematics 161(7-8), 961–969 (2013)
7. Fox, J., Pach, J., Suk, A.: The number of edges in k-quasi-planar graphs. SIAM J. Discrete Math. 27(1), 550–561 (2013)
8. Hong, S.-H., Eades, P., Liotta, G., Poon, S.-H.: Fáry's theorem for 1-planar graphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 335–346. Springer, Heidelberg (2012)
9. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. Journal of Graph Theory 72(1), 30–71 (2013)
10. Pach, J., Pinchasi, R., Sharir, M., Tóth, G.: Topological graphs with no large grids. Graphs and Combinatorics 21, 355–364 (2005)
11. Pach, J., Radoičić, R., Tóth, G.: Relaxing planarity for topological graphs. In: Akiyama, J., Kano, M. (eds.) JCDCG 2002. LNCS, vol. 2866, pp. 221–232. Springer, Heidelberg (2003)
12. Pach, J., Shahrokhi, F., Szegedy, M.: Applications of the crossing number. In: Proc. 10th Annu. ACM Sympos. Comput. Geom., pp. 198–202 (1994)
13. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. Combinatorica 17, 427–439 (1997)

Cops and Robbers on Intersection Graphs[⋆]

Tomáš Gavenčiak[1,⋆⋆], Vít Jelínek[2], Pavel Klavík[2,⋆⋆], and Jan Kratochvíl[1,⋆⋆]

[1] Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
{gavento,honza}@kam.mff.cuni.cz

[2] Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
{jelinek,klavik}@iuuk.mff.cuni.cz

Abstract. The game of cops and robber, introduced by Nowakowski and Winkler in 1983, is played by two players on a graph G, one controlling k cops and the other one robber, all positioned on V_G. The players alternate in moving their pieces to distance at most 1 each. The cops win if they capture the robber, the robber wins by escaping indefinitely. The cop-number of G, that is the smallest k such that k cops win the game, has recently been a widely studied parameter.

Intersection graph classes are defined by their geometric representations: the vertices are represented by certain geometrical shapes and two vertices are adjacent if and only if their representations intersect. Some well-known intersection classes include interval and string graphs. Various properties of many of these classes have been studied recently, including an interest in their game-theoretic properties.

In this paper we show an upper bound on the cop-number of string graphs and sharp bounds on the cop-number of interval filament graphs, circular graphs, circular arc graphs and function graphs. These results also imply polynomial algorithms determining cop-number for all these classes and their sub-classes.

Keywords: intersection graphs, string graphs, interval filament graphs, cop and robber, pursuit games, games on graphs.

1 Introduction

The Cops and Robber game on graphs has been introduced by Winkler and Nowakowski [9] and independently by Quilliot [11]. In this paper, we investigate this game on the classes of intersection graphs.

Rules of The Game. In this game two players alternate their moves. Player One (called the Cops) places k cops in the graph that is the playground of the game, i.e., the Cops chooses k vertices of the graph. Then Player Two (called the Robber) chooses a vertex. Then the players alternate. In the Cops' move,

⋆ Supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023.

⋆⋆ Supported by Charles University as GAUK 196213.

every cop either stays in its vertex or moves to one of its neighbors. More cops may occupy the same vertex if they wish so. In the Robber's move, the Robber either stays in its vertex, or goes to a neighboring vertex. The game ends if a cop occupies the same vertex as the robber, we say that the robber was captured in such a case. The *cop number* of a graph G is the minimum number k such that Player One has a winning strategy on G with k cops. For a given class of graphs, the *maximum cop number* is the maximum cop number achieved by a graph of this class (possibly $+\infty$ if unbounded).

Known Results. Graphs of the cop number one were characterized already by Quilliot [11]. These are the graphs whose vertices can be linearly ordered $v_1, v_2, \ldots, v_n$ so that each v_i is a corner of $G[v_1, \ldots, v_i]$, i.e., v_i has a neighbor v_j for $j < i$ such that v_j is adjacent to all other neighbors of v_i.

For k part of the input, deciding whether the cop number of a graph does not exceed has been shown to be NP-hard [3] and recently even PSPACE-hard [7]. It is even EXPTIME-complete to decide whether the cops can capture the robber if their initial positions (both for the cops and the robber) are given as part of the input. We can test whether k cops suffice to capture the robber on an n-vertex graph, we can search the game graph which has $O(n^{k+1})$ vertices to find a winning strategy for cops. So on the other hand if k is a fixed constant, we get a polynomial-time algorithm.

Determining bounds on the cop number for special graph classes is a recently intensively studied question. For general graphs on n vertices it is known that at least $\sqrt{n}$ cops may be needed (e.g., for the incidence graph of a finite projective plane). On the other hand, the celebrated Meyniel's conjecture states that the cop number of a connected n-vertex graph is $O(\sqrt{n})$. For more details and results see the recent book [2].

What one would like to see are large classes of graphs for which a fixed number of cops are always sufficient to capture the robber. Aigner and Fromme [1] showed that the maximum cop number of planar graphs is 3. This result has been generalized to bounded genus graphs by Quilliot [11] and Schroeder [12]. However, while for planar graphs the constant 3 is known to be the best possible, already for genus 1 the exact value of the maximum cop number of toroidal graphs is not known.

Our Results. It has been asked at several occasions, last during the Banff Workshop on Graph Searching in October 2012 whether intersection-defined graph classes (other than interval graphs) have bounded maximum cop numbers. The classes in question have included circle graphs, intersection graphs of disks in the plane, graphs of boxicity 2, and others. We solve the question in affirmative in the most general way. Before stating the result let us note that intersection graphs of arc-connected sets (i.e., intersection graphs of connected regions bounded by closed simple Jordan curves) are of the main interest, as it is known that every graph is an intersection graph of topologically connected sets in the plane. Intersection graphs of regions are exactly intersection graphs of curves in the plane, the so called *string graphs* [13]. See [8] for an overview of the topic.

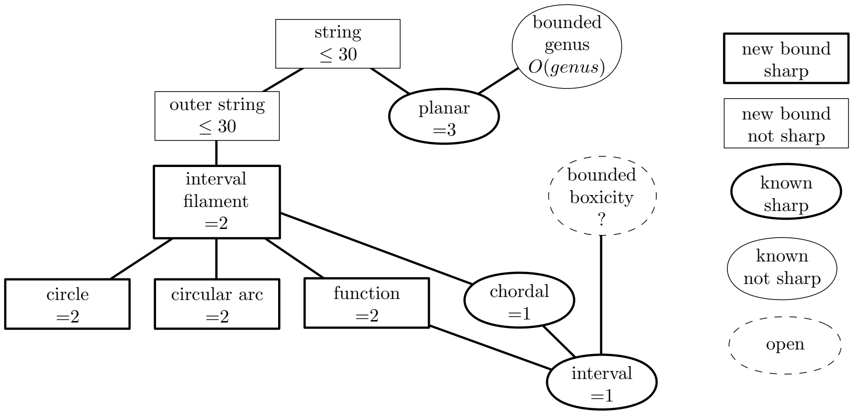


Fig. 1. Known results and new results of this paper. The Hasse diagram depicts inclusions between important classes of geometrically represented and intersection graphs.

For overview of results of this paper, see Fig. 1. Our main result is the following.

Theorem 1. *Let G be a connected string graph. Then 30 cops have a strategy to capture a robber in G.*

The constant does not seem to be optimal but we answer the question by showing that the maximum cop number of these classes are bounded. We prefer to keep the proof readable and did not try to improve on this bound much.

From other graph classes that have been questioned we single out *circle graphs*, the intersection graphs of chords of a circle. For them one cop is not enough as they contain all cycles, but we prove that two cops always suffice. We prove this in the setting of a much wider class of so called *interval filament graphs.*

Theorem 2. *Let G be a connected interval filament graph on n vertices. Then 2 cops have a strategy to capture a robber in G in $O(n)$ moves. Moreover, there exist interval filament graphs such that a robber cannot be captured with a single cop.*

We note that the strategies of cops of both theorems are geometric, based on an intersection representation of G; so without knowledge of a representation they cannot be applied. If only the graph G is given, we cannot construct these representation since recognition of both classes is known to be NP-complete [6,10]. Nevertheless, our bounds on the maximum cop number still apply and we can determine in time $\mathcal{O}(n^{k+1})$ a winning strategy for the cops using the standard algorithm searching the game space. Further, one can use our results as a rather strange heuristic for testing whether the given graph is a string graph, or an interval filament graph. If the cop-win number of a connected graph G is too high, it cannot be represented by arc-connected sets in the plane.

2 Definitions and Preliminaries

Let $G = (V, E)$ be a graph. For a vertex v, we use the *open neighborhood* $N(v) = \{u : uv \in E\}$ and the *closed neighborhood* $N[v] = N(v) \cup \{v\}$. Similarly for $V' \subseteq V$, we put $N[V'] = \bigcup_{v \in V'} N[v]$ and $N(V') = N[V'] \setminus V'$. For $V' \subseteq V$, we denote by $G|_{V'}$ the subgraph of G induced by V'.

Intersection Representations. An *intersection representation* of G is a map $\varphi : V \to 2^X$ for some ground set X such that the edges of G are prescribed by intersections of the sets $\varphi(v)$; formally, $uv \in E$ if and only if $\varphi(u) \cap \varphi(v) \neq \emptyset$. Given the (multi)set of the images of φ, the corresponding *intersection graph* is uniquely defined up to isomorphism.

The ground set X and the images of φ are usually somehow restricted to get a particulate intersection graph family. For example, to get the well-known class of interval graphs, let $X = \mathbb{R}$ and require every $\varphi(v)$ to be a closed interval.

A *string graph* is an intersection graph of a set of *strings*. We let $X = \mathbb{R}^2$ and every $\varphi(v)$ is required to be a finite curve, in this context called a string, that is a continuous image of the interval $[0, 1]$ to $\mathbb{R}^2$.

We also study a subclass of string graphs called *interval filament graphs*. An interval filament graph is a string graph such that each *interval filament* $\varphi(v)$ is a continuous (x-monotone) function defined on $[a, b]$ such that $\varphi(v)(a) = \varphi(v)(b) = 0$ and $\varphi(v)(x) > 0$ for all $x \in (a, b)$. This class was introduced by Gavril [4] as a generalisation of interval graphs on which his algorithm for weighted maximum clique/independent set can be used. The connection with interval graphs is as follows: Let $\varphi(u)$ is a filament defined on $[a, b]$ and $\varphi(v)$ be filament defined on $[c, d]$. If the intervals $[a, b]$ and $[c, d]$ are disjoint, then $\varphi(u)$ cannot intersect $\varphi(v)$. If the intervals single overlaps, e.g. $a < c < b < d$, then the filaments have to intersect. But if one interval is contained inside the other interval, e.g, $a < c < d < b$, then there the filaments $\varphi(u)$ and $\varphi(v)$ might or might not intersect. (Unlike in interval graphs where the edge/non-edge uv depends only on intersection of $[a, b]$ and $[c, d]$.)

Assumptions on Representations. Let $\varphi : V \to 2^{\mathbb{R}^2}$ be a string representation of G. Without loss of generality, we may assume that there is only a finite number of string intersections in the representation, that strings never only touch without either also crossing each other or at least one of them ending, that no three or more strings meet at the same point and that no string self-intersects. This follows from the fact that strings are compact subsets of the plane and as such can replaced by piece-wise linear curves with finite numbers of linear segments, for more details see [5]. We always assume these properties.

Regions. For a $V' \subseteq V$, let $\varphi(V') = \bigcup_{v \in V'} \varphi(v)$. Then $\varphi(V')$ divides $\mathbb{R}^2$ into one or more *open regions* which are the connected sets of $\mathbb{R}^2 \setminus \varphi(V')$. Let $\mathcal{A}(\varphi(V'))$ be the set of these regions. Since we assume there are finitely many intersections, the set $\mathcal{A}(\varphi(V'))$ is always finite throughout this paper. Let $\mathcal{C}(X)$ denote the topological closure of a set X. We mostly use this operator on elements of some $\mathcal{A}(\varphi(V'))$ to include the bounding curves.

Meetings. Let π_1 and π_2 be curves (as continuous functions from $[0, 1]$). A *meeting* of these curves are intervals $I_1, I_2 \subseteq [0, 1]$ such that $\pi_1[I_1] = \pi_2[I_2]$ and this intersection has non-zero length. π_1 and π_2 meet if they have at least one meeting. Note that one-point intersection (crossing) of curves is not a meeting.

3 Capturing Robber in Interval Filament Graphs

In this section, we establish a strategy for two cops to catch a robber in any connected interval filament graph with a given representation φ. In other words, we show that the cop-number of interval filament graphs is at most two. Since interval filament graphs contain all cycles C_n, it is easy to see that one cop is not sufficient. So we prove that two is the maximum cop number of interval filament graphs.

We have several assumptions on the representation φ. First, all filaments have only finitely many intersections. If a filament $\varphi(u)$ is defined on $[a, b]$, we call a the *left endpoint* and b the *right endpoint* of $\varphi(u)$. Another assumption is that the filaments have pairwise distinct endpoints and the defining intervals are always non-trivial.

In the description, we move the cops on the representation φ, and we say that a cop takes a filament $\varphi(u)$ if it is placed on the vertex u which this filament represents. We shall assume that the robber never moves into the neighborhood of a vertex taken by a cop, and a cop catches the robber immediately if he stands on a neighboring vertex.

Filaments and Regions. It is important that each filament splits the half-plane into two regions: the *top region* and the *bottom region*. We say that a filament $\varphi(u)$ is *nested* in a filament $\varphi(v)$ if $\varphi(u)$ is contained in the bottom region of $\varphi(v)$. We say that the robber *is/moves/stays in a region* if he is/moves/stays in a filament contained in this region. The robber is *confined* by $\varphi(u)$ if a cop takes $\varphi(u)$ and the robber is in the bottom region of $\varphi(u)$.

Lemma 3. *Suppose that the robber is confined in $\varphi(u)$. Then he stays in the bottom region of $\varphi(u)$ until the cops moves from $\varphi(u)$.*

Proof. This is obvious since to move from one region to another, the robber has to use a filament $\varphi(v)$ which crosses $\varphi(u)$. But then v is a neighbor of u, and the cop catches the robber. □

A filament $\varphi(u)$ is called *top in x* if it maximizes the value $\varphi(v)(x)$ over all filaments v defined for x. Suppose that ℓ is the left-most and r is the right-most endpoint of the representation. We have a *sequence of top filaments* $\{\varphi(t_i)\}_{i=1}^{k}$ as we traverse from ℓ to r. We note that one filament can appear several times in this sequence. See Fig. 2 for an example.

Let $\varphi(t_i)$ be top in x_i. Each filament $\varphi(t_i)$ together with the upward ray starting at $\big(x_i, \varphi(t_i)(x_i)\big)$ separates the half-plane into three regions: the *left region*, the *bottom region* and the *right region*. The key property is that there is no filament intersecting the left and right regions, and avoiding $\varphi(t_i)$.

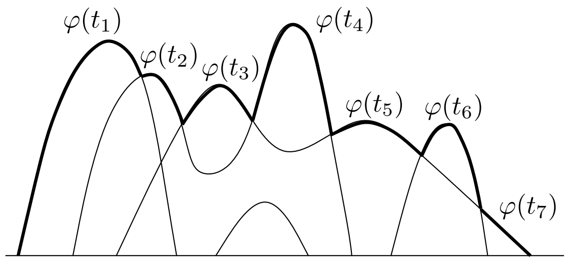


Fig. 2. An example of a sequence of top filaments. Only the top part of each $\varphi(t_i)$ is depicted in bold.

Lemma 4. *Suppose that a cop stands on $\varphi(t_i)$ and the robber is in the right region. If the cop moves to the neighboring $\varphi(t_j)$ with the maximal index j, the robber cannot move to the left region of $\varphi(t_j)$.*

Proof. Since $\varphi(t_i)$ on $[a, b]$ intersects $\varphi(t_j)$ on $[c, d]$ and j is maximal, we have $a < c < b < d$. Suppose that the cop moves from $\varphi(t_i)$ to $\varphi(t_j)$ and the robber stands on a filament $\varphi(u)$ defined on $[e, f]$. We know that $c < b < e < f$, so $\varphi(u)$ does not intersect the left region of $\varphi(t_j)$. And since $\varphi(t_j)$ is top, there is no path going to the left region which avoids $\varphi(t_j)$. So the robber cannot move there. □

Theorem 2. We are ready to prove that the maximum cop number of interval filament graphs is equal two.

Proof (Theorem 2). We already argued that two cops are necessary for some interval filament graphs. We now describe a strategy how to catch a robber with two cops. We call one cop the *guard*, and the other one the *hunter*. The guard stays on one filaments $\varphi(u)$ such that the robber is confined by it; so according Lemma 3, the robber can only move in the bottom region of $\varphi(u)$. The guard stays on $\varphi(u)$ till the robber is either caught by the hunter, or confined by the hunter in some filament $\varphi(v)$ nested in $\varphi(u)$. If the confinement happens, the guard moves to the filament $\varphi(v)$ taken by the hunter, and then the hunter proceeds with catching the robber inside the bottom region of $\varphi(v)$.

So the strategy catches the robber in phases, and each phase starts by the guard cofining the robber by taking some filament $\varphi(u)$ and ends by the guard moving to a nested filament $\varphi(v)$. In the beginning, both cops stand at any filament. For the initial phase, we can imagine that the guard takes some imaginary filament in infinity, so the robber is confined to its bottom region, i.e., to the entire graph G. The guard can move arbitrarily, or just stand at his initial position.

Suppose that we are in some phase where the guard is placed on $\varphi(u)$. Since the guard stays there till the robber is confined in some nested $\varphi(v)$, the strategy ensures that the robber can never move to or through $\varphi(u)$ and all filaments which are contained in the top region of $\varphi(u)$ or intersect $\varphi(u)$ without being cauptured. We remove these vertices from the graph and additionaly, if the resulting graph contains multiple components, we remove all of them except the one containing the robber, as the robber would have to move to one of the already removed vertices to get there, getting captured in the process. We call this a *simplified representation.* It is important that as we remove these filaments for the movements of the robber, the graph does not become disconnected and the hunter and the guard can still use them to travel towards the robber.

Let $\{\varphi(t_i)\}_{i=1}^{k}$ be the sequence of top intervals in the simplified representation. The hunter first goes to $\varphi(t_1)$. When he arrives to $\varphi(t_1)$, the robber cannot be in the left region of $\varphi(t_1)$ since there is no filament contained there. Now suppose that the hunter is in $\varphi(t_i)$ and assume the induction hypothesis that the robber is not in the left region of $\varphi(t_i)$. If the robber is confined in $\varphi(t_i)$, the phase ends with the guard moving towards $\varphi(t_i)$. If the robber is in the right region of $\varphi(t_i)$, the hunter moves to the neighbor $\varphi(t_j)$ of the maximal index j. According to Lemma 4, the robber cannot move to the left region of $\varphi(t_{i+1})$, so he is either in the bottom or the right region. The robber cannot stay in the right regions forever since $\varphi(t_k)$ has no filament contained in the right region, so after some time it is confined in $\varphi(t_i)$ or caught directly.

Since there are only finitely many filaments nested in each other, the strategy proceeds in finitely many phases and the robber is caught. With a small modification, we can prove that this strategy catches the robber in $\mathcal{O}(n)$ turns. Suppose that initially both cops are placed in the filament with the left-most endpoint ℓ.

It is sufficient to notice that in each turn only one cop moves, and if the cops use always shortest paths, they never visit any vertex of the graph more twice. To see this, suppose that both cops are on $\varphi(u)$ in the beginning of a phase. Then it takes several moves to get to the simplified representation to either $\varphi(t_1)$, or $\varphi(t_k)$, and at most k moves to get to $\varphi(t_1)$. Then the hunter confines the robber in $\varphi(t_i)$ in at most k moves, and the guard gets there in at most k moves. It is important that in further phases the cops only move in the simplified representation in the bottom region of $\varphi(t_i)$, except for the filaments used to get into the simplified representation (which are used at most two times by each cop). □

4 Guarding Paths and Curves in String Graphs

Shortest Paths. We recall a lemma by Aigner and Fromme [1] giving us a strategy to prevent the robber to enter any given shortest path using only one cop in general graphs.

Lemma 5 (Lemma 4 in [1]). *Let G be a graph, and $P = u = p_0, p_1, \ldots, p_k = v$ be any shortest $u - v$ path. Then a single cop C can, after a finite number of moves (used to move cop C to an appropriate position on P), prevent the robber from entering P. That is, if the robber ever moved on P, he would be captured in the next move.*

This result turned out to be particularly useful for planar graphs where one can cut the graph by protecting several shortest paths. For intersection graph, this is not true, and so we need a more general result. We show that in general graphs we can also protect the neighbourhood of a given shortest path using five cops. We note that this result may be of independent interest. The proof of the theorem is in the full version.

Lemma 6. *Let G be a graph, and P be any shortest $u - v$ path. Then five cops $C_{-2}, C_{-1}, C_0, C_1, C_2$ can, after a finite number of initial moves (used to move the cops to appropriate positions on P), prevent the robber from entering $N[P]$, i.e., capture the robber when he moves onto P or its neighbourhood.*

In the case five cops play as in the Lemma, we say that P is *guarded* by the five cops. In the following discussion, when we say "start protecting a path", we do not explicitly mention the initial time required to position the five cops onto the path and assume that the strategy waits for enough turns.

Shortest Curves. Since our strategy for string graphs is geometric, we introduce a geometric concept of shortest curves as particular curves through the string representation of a shortest path.

Let G be a string graph together with a fixed string representation φ, and P be any shortest $u - v$ path. Suppose that we choose two points $\pi_u \in \varphi(u)$ and $\pi_v \in \varphi(v)$. Let $\pi_{uv} \subseteq \varphi(P)$ be a curve from π_u to π_v such that for every $p \in P$ it has a connected intersection with $\phi(p)$. We call π_{uv} a *shortest curve of P* with endpoints π_u and π_v. A curve π is called a *shortest curve* if it is a shortest curve of some shortest path.

The shortest path corresponding to a shortest curve π is actually uniquely defined by the sequence of strings that intersect π on a substring of non-zero length. To *guard a shortest curve* π means to guard its corresponding shortest path. The number of its strings is the *length* of π; the geometric length of π plays no role in this paper.

Corollary 7. *Let G be a string graph together with a string representation φ and let π be a shortest curve. Then five cops can (after a finite number of initial moves) prevent the robber from entering any string intersecting π.*

Proof. Let P be the shortest path such that π is a shortest curve of P. By guarding P, the cops prevent the robber to enter $N[P]$. □

Lemma 8. *Any continuous part of a shortest curve is also a shortest curve.*

Proof. This follows from the fact that any sub-path of a shortest path is also a shortest path. □

Lemma 9. *If two shortest curves π_1 and π_2 meet more than once, part of π_2 can be re-routed via π_1 such that they meet only once.*

Proof. Every meeting implies shared vertices. Let u be the first and v the last shared vertex when going along π_2, denote π_u the start of the meeting on u and

π_v the end of the meeting on v. Then let π_2' consist of the initial part of π_2 up to π_u, then the part of π_1 between π_u and π_v, and then again π_2 from π_v onward. The length of π_2' is not larger than that of π_2 since the lengths of the $u - v$ sub-paths under π_1 and π_2 must be the same. □

5 Capturing Robber in String Graphs

In this section we show that the number of cops sufficient to capture a robber on a connected string graphs is bounded. Note that this is not the case for disconnected graphs, since at least one cop is needed for every component (as the robber could otherwise choose a cop-free component after the cops are positioned).

Before we prove Theorem 1, we define a graph restricted to a region and show how can we use a strategy for a situation in the restricted graph for a related situation in the original graph.

Given a region $B \subset \mathbb{R}^2$, let $G|_B$ be the intersection graph G' of the curves of φ *restricted to* B. This operation may remove vertices (entire strings outside B), remove edges (crossings outside B) and it also splits each vertex v whose string $\varphi(v)$ leaves and then reenters B at least once. In the last case, every arc-connected part of $\varphi(v) \cap B$ spans a new vertex v_i. The new vertices are also called the *splits* of v. The new graph is again a string graph with representation $\varphi|_B$ directly derived from φ. Note that this operation preserves the representation properties assumed above and we only use it with closed B yielding finite graphs.

Lemma 10. *If there is a cop's strategy $\mathcal{S}'$ capturing a robber in G' with a closed curve π guarded while never letting him leave $R \subseteq V_{G'}$ and G' is obtained from G by removing vertices, removing edges not ending in R and splitting vertices not in R, there is a strategy $\mathcal{S}$ for the same number of cops capturing the robber on G with π guarded while never letting him leave R.*

Proof. The strategy $\mathcal{S}$ plays out as $\mathcal{S}'$ except when $\mathcal{S}'$ wants to move a cop to a split $v_i \in V_{G'}$ of $v \in V_G$, move the cop to v. All such moves are possible and robber's choices while in R are not extended in any way. When the robber leaves R from $u \in R$ to v which is split in G', $\mathcal{S}$ plays as if the robber moved to one of the splits adjacent to r and captures him immediately. □

Proof (Theorem 1). Let φ be a string representation of G. We denote the strings entirely contained within R by $V_R = \{v \in V \mid \varphi(v) \subseteq R\}$.

We prove the following claim by induction on $|\mathcal{A}(\varphi(V))|$, that is the number of regions of the representation, with the base case $|\mathcal{A}(\varphi(V))| \leq 2$ trivially won for the cops.

Note that with some additional technical assumptions, we could use an induction-like argument on the *area* of the region S_{abc} defined by a simple closed curve π_{abc} as in the claim since the area always decreases by at least the area of the smallest region of $\mathcal{A}(\varphi(V))$ and the base case can be estabilished for regions with the area smaller than the smallest region of $\mathcal{A}(\varphi(V))$.

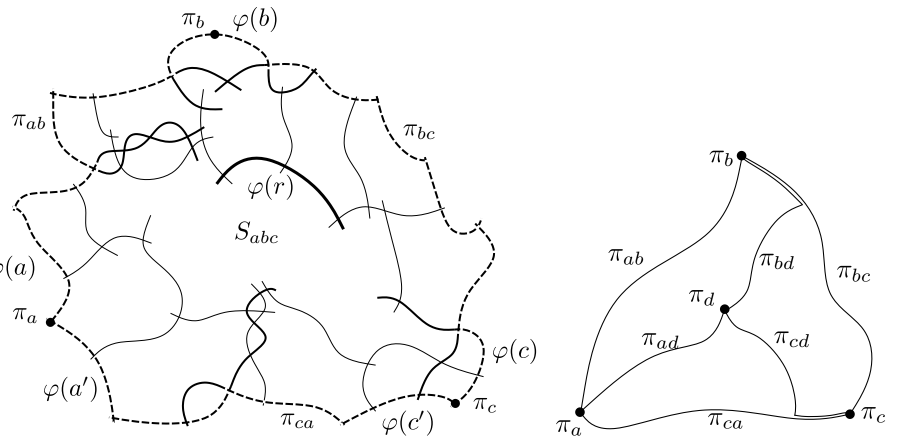


Fig. 3. Situations in the claim in proof of Theorem 1. *Left:* The claim assumptions. The shortest curves π_{ab}, π_{bc} and π_{ca} are dotted, the strings of the corresponding shortest paths P_{ab}, P_{bc} and $P_{c'a'}$ are thicker. Note that P_{ab} and P_{bc} share the vertex b while the endpoints c and c' of P_{bc} and $P_{c'a'}$ are only adjacent. *Right:* The main idea of the inductive proof. Note that the interaction of the curves can be much more complicated.

Claim. *Assume the following game state on a connected graph G with a string representation φ. The cops guard three shortest curves π_{ab}, π_{bc} and π_{ca} (with 5 cops each), the remaining 15 cops are positioned arbitrarily. The curves form a triangle with corners π_a, π_b and π_c, the curve π_{ab} goes from π_a to π_b, π_{bc} from π_b to π_c and π_{ca} from π_c to π_a. The curves are disjoint except for their endpoints. Denote $\pi_{abc} = \pi_{ab} \cup \pi_{bc} \cup \pi_{ca}$ the simple closed curve and denote the open region inside it S_{abc}. Additionally, $\varphi(V) \subseteq \mathcal{C}(S_{abc})$, that is all the strings of G are inside or touch S_{abc} in their entire length.*

The robber is on $r \in V$ with $\varphi(r)$ disjoint from π_{abc} and with $\varphi(r) \subseteq S_{abc}$. See Fig. 3 for an illustration. In this case, 30 cops have a strategy to capture the robber such that they never let the robber leave S_{abc}.

The proof of the claim is presented in the full version of this paper. The main idea is to choose $d \in V'$ and $\pi_d \in \varphi(d)$ together with shortest curves π_{ad}, π_{bd} and π_{cd} and guard them with 15 cops. Then the robber will be in one of the regions of $\mathcal{A}(\pi_{abc} \cup \pi_{ad} \cup \pi_{bd} \cup \pi_{cd})$ and we choose three sub-strings bounding this region and then use the induction assumption. See Fig. 3 for an illustration. The complication of the proof are the many ways the curves π_{ad}, π_{bd} and π_{cd} may interact.

Now we need to satisfy the conditions of the claim. If G is a tree, one cop is enough to capture the robber. Otherwise, let C be any shortest cycle in G and choose shortest curves π_{ab}, π_{bc} and π_{ca} in $\varphi(C)$ to form a simple cycle and let S_{abc} be the region inside $\pi_{abc} = \pi_{ab} \cup \pi_{bc} \cup \pi_{ca}$. Start the game with 15 cops guarding π_{ab}, π_{bc} and π_{ca}.

In case the robber is inside π_{abc}, apply the claim on $G|_{\mathcal{C}(S_{abc})}$ with π_{ab}, π_{bc} and π_{ca} guarded. In case the robber is outside π_{abc}, apply circular inversion on

φ to obtain φ' with the robber inside π'_{abc} (which is the inverted image of π_{abc} and has interior S'_{efg}) and apply the claim on $G|_{\mathcal{C}(S'_{efg})}$ with π_{ef}, π_{fg} and π_{ge} guarded and using the representation φ'. □

6 Conclusions

We showed the sharp bound on cop-number for circle, circular arc, function and interval filament graphs and a fixed upper bound for string graphs (and therefore all subclasses of string graphs). It still remains to decide whether other intersection classes have bounded cop-number, such as bounded boxiciy graphs. Boxicity k graphs are intersection graphs of axis-aligned boxes in $\mathbb{R}^k$. Another direction is improving the bounds for string graphs, outer string graphs, etc.

References

1. Aigner, M., Fromme, M.: Game of cops and robbers. Discrete Appl. Math. 8(1), 1–12 (1984)
2. Bonato, A., Nowakowski, R.J.: The Game of Cops and Robbers on Graphs. American Mathematical Society (2011)
3. Fomin, F.V., Golovach, P.A., Kratochvíl, J., Nisse, N., Suchan, K.: Pursuing a fast robber on a graph. Theor. Comput. Sci. 411(7-9), 1167–1181 (2010)
4. Gavril, F.: Maximum weight independent sets and cliques in intersection graphs of filaments. Inf. Process. Lett. 73(5-6), 181–188 (2000)
5. Kratochvíl, J., Goljan, M., Kučera, P.: String graphs. Rozpravy ČSAV.: Řada matem. a přírodních věd, Academia (1986)
6. Kratochvíl, J.: String graphs. II. recognizing string graphs is NP-hard. Journal of Combinatorial Theory, Series B 52(1), 67–78 (1991)
7. Mamino, M.: On the computational complexity of a game of cops and robbers. Theor. Comput. Sci. 477, 48–56 (2013)
8. McKee, T., McMorris, F.: Topics in Intersection Graph Theory. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (1999)
9. Nowakowski, R., Winkler, P.: Vertex-to-vertex pursuit in a graph. Discrete Math. 43, 235–239 (1983)
10. Pergel, M.: Recognition of polygon-circle graphs and graphs of interval filaments is NP-complete. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 238–247. Springer, Heidelberg (2007)
11. Quilliot, A.: Some results about pursuit games on metric spaces obtained through graph theory techniques. European J. Combin. 7, 55–66 (1986)
12. Schroeder, B.S.W.: The copnumber of a graph is bounded by 3/2 genus(g) + 3. Trends Math., pp. 243–263. Birkhäuser, Boston (2001)
13. Sinden, F.W.: Topology of thin film RC-circuits. Bell System Technical Journal 45, 1639–1662 (1966)

SEFE with No Mapping via Large Induced Outerplane Graphs in Plane Graphs

Patrizio Angelini[1], William Evans[2], Fabrizio Frati[3], and Joachim Gudmundsson[3]

[1] Dipartimento di Ingegneria, Roma Tre University, Italy
angelini@dia.uniroma3.it

[2] Department of Computer Science, University of British Columbia, Canada
will@cs.ubc.ca

[3] School of Information Technologies, The University of Sydney, Australia
brillo@it.usyd.edu.au
joachim.gudmundsson@sydney.edu.au

Abstract. We show that every n-vertex planar graph admits a simultaneous embedding with no mapping and with fixed edges with any $(n/2)$-vertex planar graph. In order to achieve this result, we prove that every n-vertex plane graph has an induced outerplane subgraph containing at least $n/2$ vertices. Also, we show that every n-vertex planar graph and every n-vertex planar partial 3-tree admit a simultaneous embedding with no mapping and with fixed edges.

1 Introduction

Simultaneous embedding is a flourishing area of research studying topological and geometric properties of planar drawings of multiple graphs on the same point set. In the seminal paper in the area by Braß *et al.* [7], two types of simultaneous embedding are defined, namely *with mapping* and *with no mapping*. In the former variant, a bijective mapping between the vertex sets of any two graphs G_1 and G_2 to be drawn is part of the problem's input, and the goal is to construct a planar drawing of G_1 and a planar drawing of G_2 so that corresponding vertices are mapped to the same point. In the latter variant, the drawing algorithm is free to map any vertex of G_1 to any vertex of G_2 (still the n vertices of G_1 and the n vertices of G_2 have to be placed on the same n points). Simultaneous embeddings have been studied with respect to two different drawing standards: In *geometric simultaneous embedding*, edges are required to be straight-line segments. In *simultaneous embedding with fixed edges* (also known as SEFE), edges can be arbitrary Jordan curves, but each edge that belongs to two graphs G_1 and G_2 has to be represented by the same Jordan curve in the drawing of G_1 and in the drawing of G_2.

Many papers deal with the problem of constructing geometric simultaneous embeddings and simultaneous embeddings with fixed edges of pairs of planar graphs in the variant *with mapping*. Typical considered problems include: (i) determining notable classes of planar graphs that always or not always admit a simultaneous embedding; (ii) designing algorithms for constructing simultaneous embeddings within small area and with few bends on the edges; (iii) determining the time complexity of testing the existence of a simultaneous embedding for a given set of graphs. We refer the reader to the recent survey by Blasiüs, Kobourov, and Rutter [4].

In contrast to the large number of papers dealing with simultaneous embedding *with mapping*, little progress has been made on the *no mapping* version of the problem.

Braß *et al.* [7] showed that any planar graph admits a geometric simultaneous embedding with no mapping with any number of outerplanar graphs. They left open the following attractive question: Do every two n-vertex planar graphs admit a geometric simultaneous embedding with no mapping?

In this paper we initiate the study of simultaneous embeddings with fixed edges and no mapping (SEFENOMAP). In this setting, the natural counterpart of the Braß *et al.* [7] question reads as follows: Do every two n-vertex planar graphs admit a SEFENOMAP ?

Since answering this question seems to be an elusive goal, we tackle the following generalization of the problem: What is the largest $k \leq n$ such that every n-vertex planar graph and every k-vertex planar graph admit a SEFENOMAP ? That is: What is the largest $k \leq n$ such that every n-vertex planar graph G_1 and every k-vertex planar graph G_2 admit two planar drawings Γ_1 and Γ_2 with their vertex sets mapped to point sets P_1 and P_2, respectively, so that $P_2 \subseteq P_1$ and so that if edges e_1 of G_1 and e_2 of G_2 have their end-vertices mapped to the same two points p_a and p_b, then e_1 and e_2 are represented by the same Jordan curve in Γ_1 and in Γ_2? We prove that $k \geq n/2$:

Theorem 1. *Every n-vertex planar graph and every $(n/2)$-vertex planar graph have a* SEFENOMAP .

Observe that the previous theorem would be easily proved if $n/2$ were replaced with $n/4$: First, consider an $(n/4)$-vertex independent set I of any n-vertex planar graph G_1 (which always exists, as a consequence of the four color theorem [11,12]). Then, construct any planar drawing Γ_1 of G_1, and let $P(I)$ be the point set on which the vertices of I are mapped in Γ_1. Finally, construct a planar drawing Γ_2 of any $(n/4)$-vertex planar graph G_2 on point set $P(I)$ (e.g. using Kaufmann and Wiese's technique [10]). Since I is an independent set, any bijective mapping between the vertex set of G_2 and I ensures that G_1 and G_2 share no edges. Thus, Γ_1 and Γ_2 are a SEFENOMAP of G_1 and G_2.

In order to get the $n/2$ bound, we study the problem of finding a large induced outerplane graph in a plane graph. A *plane graph* is a planar graph together with a *plane embedding*, that is, an equivalence class of planar drawings, where two planar drawings Γ_1 and Γ_2 are equivalent if: (1) each vertex has the same *rotation scheme* in Γ_1 and in Γ_2, i.e., the same clockwise order of the edges incident to it; (2) each face has the same *facial cycles* in Γ_1 and in Γ_2, i.e., it is delimited by the same set of cycles; and (3) Γ_1 and Γ_2 have the same *outer face*. An *outerplane graph* is a graph together with an *outerplane embedding*, that is a plane embedding where all the vertices are incident to the outer face. An *outerplanar graph* is a graph that admits an outerplane embedding; a plane embedding of an outerplanar graph is not necessarily outerplane. Consider a plane graph G and a subset V' of its vertex set. The *induced plane graph* $G[V']$ is the subgraph of G induced by V' together with the plane embedding *inherited* from G, i.e., the embedding obtained from the plane embedding of G by removing all the vertices and edges not in $G[V']$. We show the following result:

Theorem 2. *Every n-vertex plane graph $G(V, E)$ has a vertex set $V' \subseteq V$ with $|V'| \geq n/2$ such that $G[V']$ is an outerplane graph.*

Theorem 2 and the results of Gritzmann *et al.* [8] yield a proof of Theorem 1, as follows:

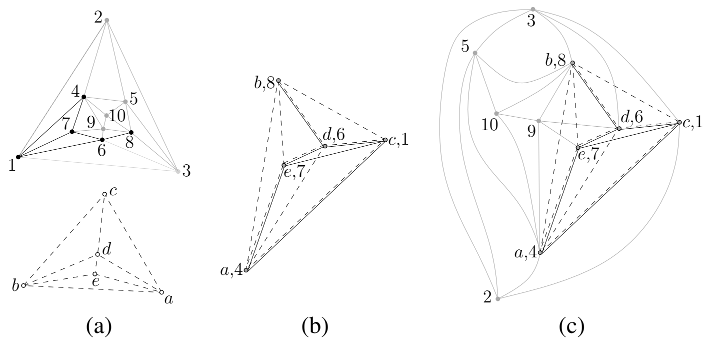


Fig. 1. (a) A 10-vertex planar graph G_1 (solid lines) and a 5-vertex planar graph G_2 (dashed lines). A 5-vertex induced outerplane graph $G_1[V']$ in G_1 is colored black. Vertices and edges of G_1 not in $G_1[V']$ are colored gray. (b) A straight-line planar drawing $\Gamma(G_2)$ of G_2 with no three collinear vertices, together with a straight-line planar drawing of $G_1[V']$ on the point set P_2 defined by the vertices of G_2 in $\Gamma(G_2)$. (c) A SEFENOMAP of G_1 and G_2.

Proof of Theorem 1: Consider any n-vertex plane graph G_1 and any $(n/2)$-vertex plane graph G_2 (see Fig. 1(a)). Let $\Gamma(G_2)$ be any straight-line planar drawing of G_2 in which no three vertices are collinear. Denote by P_2 the set of $n/2$ points to which the vertices of G_2 are mapped in $\Gamma(G_2)$. Consider any vertex subset $V' \subseteq V(G_1)$ such that $G_1[V']$ is an outerplane graph. Such a set exists by Theorem 2. Construct a straight-line planar drawing $\Gamma(G_1[V'])$ of $G_1[V']$ in which its vertices are mapped to P_2 so that the resulting drawing has the same (outerplane) embedding as $G_1[V']$. Such a drawing exists by results of Gritzmann *et al.* [8]; also it can found efficiently by results of Bose [6] (see Fig. 1(b)). Construct any planar drawing $\Gamma(G_1)$ of G_1 in which the drawing of $G_1[V']$ is $\Gamma(G_1[V'])$. Such a drawing exists, given that $\Gamma(G_1[V'])$ is a planar drawing of a plane subgraph $G_1[V']$ of G_1 preserving the embedding of $G_1[V']$ in G_1 (see Fig. 1(c)). Both $\Gamma(G_1)$ and $\Gamma(G_2)$ are planar, by construction. Also, the only edges that are possibly shared by G_1 and G_2 are those between two vertices that are mapped to P_2. However, such edges are drawn as straight-line segments both in $\Gamma(G_1)$ and in $\Gamma(G_2)$. Thus, $\Gamma(G_1)$ and $\Gamma(G_2)$ are a SEFENOMAP of G_1 and G_2. □

By the standard observation that the vertices in the odd (or even) levels of a breadth-first search tree of a planar graph induce an *outerplanar* graph, we know that G has an induced outerplanar graph with at least $n/2$ vertices. However, since its embedding in G may not be outerplane, this seems insufficient to prove the existence of a SEFENOMAP of every n-vertex and every $(n/2)$-vertex planar graph.

Theorem 2 might be of independent interest, as it is related to (in fact it is a weaker version of) one of the most famous and long-standing graph theory conjectures:

Conjecture 1. (Albertson and Berman 1979 [2]) Every n-vertex planar graph $G(V, E)$ has a vertex set $V' \subseteq V$ with $|V'| \geq n/2$ such that $G[V']$ is a forest.

Conjecture 1 would prove the existence of an $(n/4)$-vertex independent set in a planar graph without using the four color theorem [11,12]. The best known partial result

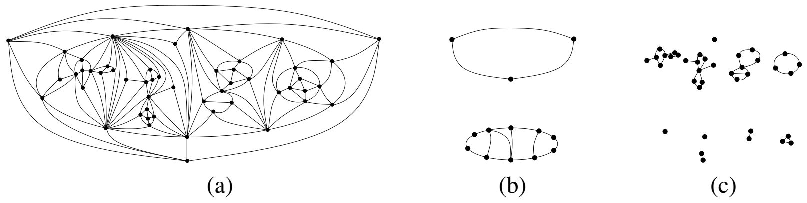


Fig. 2. (a) A maximal plane graph G with outerplanarity 4. (b) Graphs $G[V_1]$ (on the top) and $G[V_2]$ (on the bottom). (c) Graphs $G[V_3]$ (on the top) and $G[V_4]$ (on the bottom).

related to Conjecture 1 is that every planar graph has a vertex subset with $2/5$ of its vertices inducing a forest, which is a consequence of the *acyclic* 5-*colorability* of planar graphs [5]. Variants of the conjecture have also been studied such that the planar graph in which the induced forest has to be found is bipartite [1], or is outerplanar [9], or such that each connected component of the induced forest is required to be a path [14,15].

The topological structure of an outerplane graph is arguably much closer to that of a forest than the one of a non-outerplane graph. Thus the importance of Conjecture 1 may justify the study of induced outerplane graphs in plane graphs in its own right.

To complement the results of the paper, we also show the following:

Theorem 3. *Every n-vertex planar graph and every n-vertex planar partial 3-tree have a* SEFENOMAP .

Because of space limitations, some proofs are omitted or sketched. Complete proofs can be found in the full version of the paper [3].

2 Proof of Theorem 2

In this section we prove Theorem 2. We assume that the input graph G is a *maximal* plane graph, that is, a plane graph such that no edge can be added to it while maintaining planarity. In fact, if G is not maximal, then dummy edges can be added to it in order to make it a maximal plane graph G'. Then, the vertex set V' of an induced outerplane graph $G'[V']$ in G' induces an outerplane graph in G, as well.

Let $G_1^* = G$ and, for any $i \geq 1$, let G_{i+1}^* be the plane graph obtained by removing from G_i^* the set V_i of vertices incident to the outer face of G_i^* and their incident edges. Vertex set V_i is the *i-th outerplane level* of G. Denote by k the maximum index such that V_k is non-empty; then k is the *outerplanarity* of G. For any $1 \leq i \leq k$, graph $G[V_i]$ is a (not necessarily connected) outerplane graph and graph G_i^* is a (not necessarily connected) *internally-triangulated* plane graph, that is, a plane graph whose internal faces are all triangles. See Fig. 2. For $1 \leq i \leq k$, denote by $H_{i,1}^*, \ldots, H_{i,h_i}^*$ the connected components of G_i^* and, for $1 \leq j \leq h_i$, denote by $H_{i,j}$ the outerplane graph induced by the vertices incident to the outer face of $H_{i,j}^*$. Since G is maximal, for any $1 \leq i \leq k$ and for any internal face f of $G[V_i]$, at most one connected component of G_{i+1}^* lies inside f.

A 2-*coloring* $\psi = (W^*, B^*)$ of a graph H^* is a partition of the vertex set $V(H^*)$ into two sets W^* and B^*. We say that the vertices in W^* are *white* and the ones in

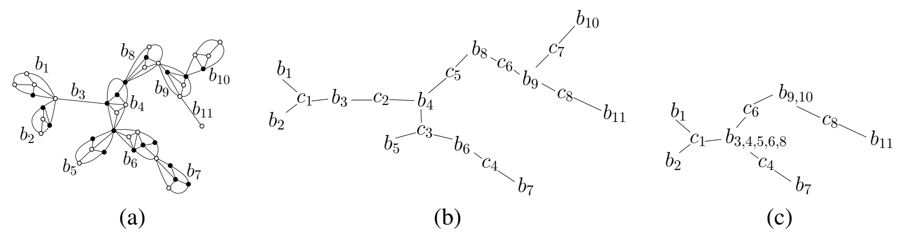


Fig. 3. (a) A connected internally-triangulated plane graph H^* with a 2-coloring ψ, (b) the block-cutvertex tree $\mathcal{BC}(H^*)$, and (c) the contracted block-cutvertex tree $\mathcal{CBC}(H^*, \psi)$

B^* are *black*. Given a 2-coloring $\psi = (W^*, B^*)$ of a plane graph H^*, the subgraph $H^*[W^*]$ of H^* is *strongly outerplane* if it is outerplane and it contains no black vertex inside any of its internal faces. We define the *surplus* of ψ as $s(H^*, \psi) = |W^*| - |B^*|$.

A *cutvertex* in a connected graph H^* is a vertex whose removal disconnects H^*. A *maximal* 2*-connected component* of H^*, also called a *block* of H^*, is an induced subgraph $H^*[V']$ of H^* such that $H^*[V']$ is 2-connected and there exists no $V'' \subseteq V(H^*)$ where $V' \subset V''$ and $H^*[V'']$ is 2-connected. The *block-cutvertex tree* $\mathcal{BC}(H^*)$ of H^* is a tree that represents the arrangement of the blocks of H^* (see Figs. 3(a) and 3(b)). Namely, $\mathcal{BC}(H^*)$ contains a *$\mathcal{B}$-node* for each block of H^* and a *$\mathcal{C}$-node* for each cutvertex of H^*; further, there is an edge between a $\mathcal{B}$-node b and a $\mathcal{C}$-node c if c is a vertex of b. Given a 2-coloring $\psi = (W^*, B^*)$ of H^*, the *contracted block-cutvertex tree* $\mathcal{CBC}(H^*, \psi)$ of H^* is the tree obtained from $\mathcal{BC}(H^*)$ by identifying all the $\mathcal{B}$-nodes that are adjacent to the same black cut-vertex c, and by removing c and its incident edges (see Fig. 3(c)). Each node of $\mathcal{CBC}(H^*, \psi)$ is either a $\mathcal{C}$-node c or a $\mathcal{BU}$-node b. In the former case, c corresponds to a white $\mathcal{C}$-node in $\mathcal{BC}(H^*)$. In the latter case, b corresponds to a maximal connected subtree $\mathcal{BC}(H^*(b))$ of $\mathcal{BC}(H^*)$ only containing $\mathcal{B}$-nodes and black $\mathcal{C}$-nodes. The *subgraph $H^*(b)$ of H^* associated with a $\mathcal{BU}$-node b* is the union of the blocks of H^* corresponding to $\mathcal{B}$-nodes in $\mathcal{BC}(H^*(b))$. Finally, we denote by $H(b)$ the outerplane graph induced by the vertices incident to the outer face of $H^*(b)$. We have the following:

Lemma 1. *For any $1 \leq i \leq k$ and any $1 \leq j \leq h_i$, there exists a 2-coloring $\psi = (W^*_{i,j}, B^*_{i,j})$ of $H^*_{i,j}$ such that:*

(1) the subgraph $H^*_{i,j}[W^*_{i,j}]$ of $H^*_{i,j}$ induced by $W^*_{i,j}$ is strongly outerplane; and

(2) for any $\mathcal{BU}$-node b in $\mathcal{CBC}(H^*_{i,j}, \psi)$, one of the following holds:
- *(a) $s(H^*_{i,j}(b), \psi) \geq |W^*_{i,j} \cap V(H_{i,j}(b))| + 1$;*
- *(b) $s(H^*_{i,j}(b), \psi) = |W^*_{i,j} \cap V(H_{i,j}(b))|$ and there exists an edge with white end-vertices incident to the outer face of $H^*_{i,j}(b)$; or*
- *(c) $s(H^*_{i,j}(b), \psi) = 1$ and $H^*_{i,j}(b)$ is a single vertex.*

Lemma 1 implies Theorem 2 as follows: Since G is a maximal plane graph, G^*_1 has one 2-connected component, hence $H^*_{1,1}(b) = H^*_{1,1} = G^*_1 = G$. By Lemma 1, there exists a 2-coloring $\psi = (W, B)$ of G such that $G[W]$ is an outerplane graph and $|W| - |B| \geq |W \cap V_1| \geq 0$, hence $|W| \geq n/2$.

We emphasize that Lemma 1 shows the existence of a large induced subgraph $H^*_{i,j}[W^*_{i,j}]$ of $H^*_{i,j}$ satisfying an even stronger property than just being outerplane; namely, the 2-coloring $\psi = (W^*_{i,j}, B^*_{i,j})$ is such that $H^*_{i,j}[W^*_{i,j}]$ is outerplane and contains no vertex belonging to $B^*_{i,j}$ in any of its internal faces.

In order to prove Lemma 1, we start by showing some sufficient conditions for a 2-coloring to induce a strongly outerplane graph in $H^*_{i,j}$. We first state a lemma arguing that a 2-coloring ψ of $H^*_{i,j}$ satisfies Condition (1) of Lemma 1 if and only if it satisfies the same condition "inside each internal face" of $H_{i,j}$. For any face f of $H_{i,j}$, we denote by C_f the cycle delimiting f; also, we denote by $H^*_{i,j}[W^*_{i,j}(f)]$ the subgraph of $H^*_{i,j}$ induced by the white vertices inside or belonging to C_f.

Lemma 2. *Let $\psi = (W^*_{i,j}, B^*_{i,j})$ be a 2-coloring of $H^*_{i,j}$. Assume that, for each internal face f of $H_{i,j}$, graph $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane. Then, $H^*_{i,j}[W^*_{i,j}]$ is strongly outerplane.*

An internal face f of $H_{i,j}$ is *empty* if it contains no vertex of G^*_{i+1} in its interior. Also, for a 2-coloring ψ of $H^*_{i,j}$, an internal face f of $H_{i,j}$ is *trivial* if it contains in its interior a connected component $H^*_{i+1,k}$ of G^*_{i+1} that is a single white vertex or such that all the vertices incident to the outer face of $H^*_{i+1,k}$ are black. We have the following.

Lemma 3. *Let $\psi = (W^*_{i,j}, B^*_{i,j})$ be a 2-coloring of $H^*_{i,j}$ and let f be a trivial face of $H_{i,j}$. Let $H^*_{i+1,k}$ be the connected component of G^*_{i+1} in f's interior. If $H^*_{i+1,k}[W^*_{i,j}]$ is strongly outerplane and if C_f contains at least one black vertex, then $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane.*

We now prove Lemma 1 by induction on the outerplanarity of $H^*_{i,j}$.

In the base case, the outerplanarity of $H^*_{i,j}$ is 1; then, color white all the vertices of $H^*_{i,j}$. Since the outerplanarity of $H^*_{i,j}$ is 1, then $H^*_{i,j}[W^*_{i,j}] = H^*_{i,j}$ is an outerplane graph, thus satisfying Condition (1) of Lemma 1. Also, consider any $\mathcal{BU}$-node b in the contracted block-cutvertex tree $\mathcal{CBC}(H^*_{i,j}, \psi)$ (which coincides with the block-cutvertex tree $\mathcal{BC}(H^*_{i,j})$, given that all the vertices of $H^*_{i,j}$ are white). All the vertices of $H^*_{i,j}(b)$ are white, hence either Condition (2b) or Condition (2c) of Lemma 1 is satisfied, depending on whether $H^*_{i,j}(b)$ has or does not have an edge, respectively.

In the inductive case, the outerplanarity of $H^*_{i,j}$ is greater than 1.

First, we inductively construct a 2-coloring $\psi_k = (W^*_{i+1,k}, B^*_{i+1,k})$, satisfying the conditions of Lemma 1, of each connected component $H^*_{i+1,k}$ of G^*_{i+1}, for $1 \leq k \leq h_{i+1}$. The 2-coloring ψ of $H^*_{i,j}$ is such that each connected component $H^*_{i+1,k}$ of G^*_{i+1} that lies inside an internal face of $H_{i,j}$ "maintains" the coloring ψ_k, i.e., a vertex of $H^*_{i+1,k}$ is white in ψ if and only if it is white in ψ_k. Then, in order to determine ψ, it suffices to describe how to color the vertices of $H_{i,j}$.

Second, we look at the internal faces of $H_{i,j}$ one at a time. When we look at a face f, we determine a set B_f of vertices of C_f that are colored black. This is done in such a way that the graph $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane even if we color white all the vertices in $V(C_f) \setminus B_f$. By Lemma 2, a 2-coloring of $H^*_{i,j}$ such that $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane for every internal face f of $H_{i,j}$ is such that $H^*_{i,j}[W^*_{i,j}]$ is strongly outerplane. We remark that, when a set B_f of vertices of C_f are colored black, the vertices in $V(C_f) \setminus B_f$ are not necessarily colored white, as a vertex in $V(C_f) \setminus B_f$

might belong to the set $B_{f'}$ of vertices that are colored black for a face $f' \neq f$ of $H_{i,j}$. In fact, only after the set B_f of vertices of C_f are colored black for *every* internal face f of $H_{i,j}$, are the remaining uncolored vertices in $H_{i,j}$ colored white.

We now describe in more detail how to color the vertices of $H_{i,j}$. We show an algorithm, that we call *algorithm cycle-breaker*, that associates a set B_f to each internal face f of $H_{i,j}$ as follows.

Empty faces: For any empty face f of $H_{i,j}$, let $B_f = \emptyset$.

Trivial faces: While there exists a vertex $v^*_{1,2}$ incident to two trivial faces f_1 and f_2 of $H_{i,j}$ to which no sets B_{f_1} and B_{f_2} have been associated yet, respectively, let $B_{f_1} = B_{f_2} = \{v^*_{1,2}\}$. When no such vertex exists, for any trivial face f of $H_{i,j}$ to which no set B_f has been associated yet, let v be any vertex of C_f and let $B_f = \{v\}$.

Non-trivial non-empty faces: Consider any non-trivial non-empty internal face f of $H_{i,j}$. Denote by $H^*_{i+1,k}$ the connected component of G^*_{i+1} inside f. By induction, for any $\mathcal{BU}$-node b in the contracted block-cutvertex tree $\mathcal{CBC}(H^*_{i+1,k}, \psi_k)$, it holds $s(H^*_{i+1,k}(b), \psi_k) \geq |W^*_{i+1,k} \cap V(H_{i+1,k}(b))| + 1$, or $s(H^*_{i+1,k}(b), \psi_k) = |W^*_{i+1,k} \cap V(H_{i+1,k}(b))|$ and there exists an edge incident to the outer face of $H^*_{i+1,k}(b)$ whose both end-vertices are white.

We repeatedly perform the following actions: (i) We pick any $\mathcal{BU}$-node b that is a leaf in $\mathcal{CBC}(H^*_{i+1,k}, \psi_k)$; (ii) we insert some vertices of C_f in B_f, based on the structure and the coloring of $H^*_{i+1,k}(b)$; and (iii) we remove b from $\mathcal{CBC}(H^*_{i+1,k}, \psi_k)$, possibly also removing its adjacent cutvertex, if it has degree one. We describe in more detail action (ii).

For every white vertex u incident to the outer face of $H^*_{i+1,k}(b)$, we define the *rightmost neighbor* $r(u,b)$ *of* u *in* C_f *from* b as follows. Denote by u' the vertex following u in the clockwise order of the vertices along the cycle delimiting the outer face of $H^*_{i+1,k}(b)$. Vertex $r(u,b)$ is the vertex preceding u' in the clockwise order of the neighbors of u. Observe that, since $H^*_{i,j}$ is internally-triangulated, then $r(u,b)$ belongs to C_f. Also, $r(u,b)$ is well-defined because u is not a cutvertex (in fact, it might be a cutvertex of $H^*_{i+1,k}$, but it is not a cutvertex of $H^*_{i+1,k}(b)$, since such a graph contains no white cut-vertex).

Suppose that $s(H^*_{i+1,k}(b), \psi_k) \geq |W^*_{i+1,k} \cap V(H_{i+1,k}(b))| + 1$. Then, for every white vertex u incident to the outer face of $H^*_{i+1,k}(b)$, we add $r(u,b)$ to B_f.

Suppose that $s(H^*_{i+1,k}(b), \psi_k) = |W^*_{i+1,k} \cap V(H_{i+1,k}(b))|$ and there exists an edge (v,v') incident to the outer face of $H^*_{i+1,k}(b)$ such that v and v' are white. Assume, w.l.o.g., that v' follows v in the clockwise order of the vertices along the cycle delimiting the outer face of $H^*_{i+1,k}(b)$. Then, for every white vertex $u \neq v$ incident to the outer face of $H^*_{i+1,k}(b)$, we add $r(u,b)$ to B_f.

After the execution of algorithm cycle-breaker, a set B_f has been defined for every internal face f of $H_{i,j}$. Then, color black all the vertices in $\bigcup_f B_f$, where the union is over all the internal faces f of $H_{i,j}$. Also, color white all the vertices of $H_{i,j}$ that are not colored black. Denote by $\psi = (W^*_{i,j}, B^*_{i,j})$ the resulting coloring of $H^*_{i,j}$. We have the following lemma, that completes the induction, and hence the proof of Lemma 1.

Lemma 4. *Coloring ψ satisfies Conditions (1) and (2) of Lemma 1.*

Proof sketch: *Condition (1).* By Lemma 2, it suffices to prove that, for every internal face f of $H_{i,j}$, graph $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane. This is trivially true for any *empty face* f of $H_{i,j}$. By construction, for any *trivial face* f of $H_{i,j}$, there is a black vertex in C_f; hence, by Lemma 3, graph $H^*_{i,j}[W^*_{i,j}(f)]$ is strongly outerplane. Consider any *non-empty non-trivial internal face* f of $H_{i,j}$ containing a connected component $H^*_{i+1,k}$ of G^*_{i+1} in its interior. Suppose, for a contradiction, that $H^*_{i,j}[W^*_{i,j}(f)]$ contains a cycle C with a vertex x in its interior. By induction and since algorithm cycle-breaker inserts in B_f at least one vertex of C_f, it follows that C contains vertices of C_f *and* vertices in the interior of C_f. Then, consider a maximal path P in $C \cap C_f$ connecting two vertices u and v; let u' be the neighbor of u in $C \setminus P$, where u', u, and v appear in this clockwise order along C, and let (u', u'') be the first edge of $H^*_{i+1,k}$ following (u', u) in the clockwise order of the edges incident to u'. Then, algorithm cycle-breaker inserts in B_f either the rightmost neighbor $r(u', b)$ of u' in C_f from a node b of $\mathcal{CBC}(H^*_{i+1,k}, \psi_k)$, or the rightmost neighbor $r(u'', b')$ of u'' in C_f from a node b' of $\mathcal{CBC}(H^*_{i+1,k}, \psi_k)$. By planarity, $r(u', b)$ and $r(u'', b')$ belong to P, thus contradicting the fact that all the vertices of P are white.

Condition (2). Consider any $\mathcal{BU}$-node b in $\mathcal{CBC}(H^*_{i,j}, \psi)$. Denote by $H_{i,j}(b)$ the outerplane graph induced by the vertices incident to the outer face of $H^*_{i,j}(b)$. The surplus $s(H^*_{i,j}(b), \psi)$ is the sum of the surpluses $s(H^*_{i+1,k}, \psi)$ of the connected components $H^*_{i+1,k}$ of G^*_{i+1} inside the internal faces of $H_{i,j}(b)$, plus the number $|W^*_{i,j} \cap V(H_{i,j}(b))|$ of white vertices in $H_{i,j}(b)$, minus the number $|B^*_{i,j} \cap V(H_{i,j}(b))|$ of black vertices in $H_{i,j}(b)$, which is equal to $|\bigcup_f B_f|$, where the union is over all the internal faces f of $H_{i,j}(b)$. The proof distinguishes three cases. In *Case A*, $H_{i,j}(b)$ contains at least one non-trivial non-empty internal face. Then, the trivial faces of $H_{i,j}(b)$ do not give any contribution to $s(H^*_{i,j}(b), \psi)$. Further, for every non-trivial non-empty internal face f of $H_{i,j}(b)$ containing a connected component $H^*_{i+1,k}$ of G^*_{i+1} in its interior, algorithm cycle-breaker inserts into B_f at most $s(H^*_{i+1,k}, \psi) - 1$ vertices. Since $H_{i,j}(b)$ contains at least one non-trivial non-empty internal face, then $s(H^*_{i,j}(b), \psi) \geq |W^*_{i,j} \cap V(H_{i,j}(b))| + 1$, thus Condition (2a) is satisfied. In *Case B*, all the faces of $H_{i,j}(b)$ are either trivial or empty, and there exists a vertex incident to two trivial faces of $H_{i,j}(b)$. Then, the sum of the surpluses $s(H^*_{i+1,k}, \psi)$ of the connected components $H^*_{i+1,k}$ of G^*_{i+1} inside the internal faces f of $H_{i,j}(b)$ is at least one greater than the number of vertices inserted by algorithm cycle-breaker into $|\bigcup_f B_f|$. Hence, $s(H^*_{i,j}(b), \psi) \geq |W^*_{i,j} \cap V(H_{i,j}(b))| + 1$, thus Condition (2a) is satisfied. Finally, in *Case C*, all the faces of $H_{i,j}(b)$ are either trivial or empty, and there exists no vertex incident to two trivial faces of $H_{i,j}(b)$. Then, the sum of the surpluses $s(H^*_{i+1,k}, \psi)$ of the connected components $H^*_{i+1,k}$ of G^*_{i+1} inside the internal faces f of $H_{i,j}(b)$ might not be greater than the number of vertices inserted by algorithm cycle-breaker into $|\bigcup_f B_f|$. However, there exists an edge incident to the outer face of $H^*_{i,j}(b)$ whose end-vertices belong to $W^*_{i,j}$, thus Condition (2b) is satisfied. □

3 Proof of Theorem 3

In this section we prove Theorem 3. It suffices to prove Theorem 3 for an n-vertex *maximal* plane graph G_1 and an n-vertex (*maximal*) plane 3-tree G_2. In fact, if G_1 and G_2 are not maximal, then they can be augmented to an n-vertex maximal plane graph G'_1 and an n-vertex plane 3-tree G'_2, respectively; the latter augmentation can be always performed, as proved in [13]. Then, a SEFENOMAP can be constructed for

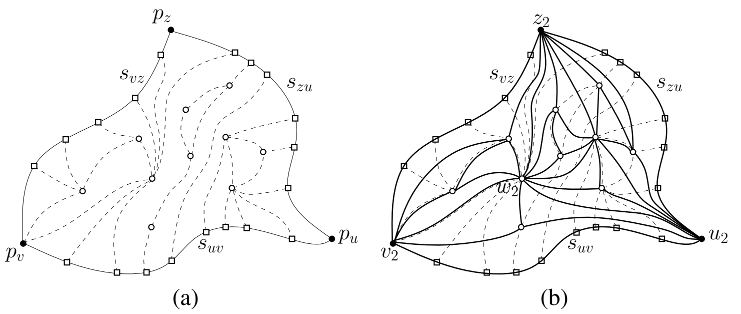


Fig. 4. (a) Setting for Lemma 5. White circles are points in P. White squares are points in R. Dashed curves are in S. Curves s_{uv}, s_{vz}, and s_{zu} are solid thin curves. (b) A planar drawing of G_2 (solid thick lines) satisfying the properties of Lemma 5.

G'_1 and G'_2, and finally the edges not in G_1 and G_2 can be removed, thus obtaining a SEFENOMAP of G_1 and G_2. In the following we assume that G_1 and G_2 are an n-vertex maximal plane graph and an n-vertex plane 3-tree, respectively, for some $n \geq 3$. Denote by $C_i = (u_i, v_i, z_i)$ the cycle delimiting the outer face of G_i, for $i = 1, 2$, where vertices u_i, v_i, and z_i appear in this clockwise order along C_i.

Let p_u, p_v, and p_z be three points in the plane. Let s_{uv}, s_{vz}, and s_{zu} be three curves connecting p_u and p_v, connecting p_v and p_z, and connecting p_z and p_u, respectively, that do not intersect except at their common end-points. Let Δ_{uvz} be the closed curve $s_{uv} \cup s_{vz} \cup s_{zu}$. Assume that p_u, p_v, and p_z appear in this clockwise order along Δ_{uvz}. Denoting by $int(\Delta)$ the interior of a closed curve Δ, let $cl(\Delta) = int(\Delta) \cup \Delta$. Let P be a set of $n - 3 \geq 0$ points in $int(\Delta_{uvz})$ and let R be a set of points on Δ_{uvz}, where $p_u, p_v, p_z \in R$. Let S be a set of curves whose end-points are in $R \cup P$ such that: (i) No two curves in S intersect, except possibly at common end-points, (ii) no two curves in S connect the same pair of points in $R \cup P$, (iii) each curve in S is contained in $cl(\Delta_{uvz})$, (iv) any point in R, except possibly for p_u, p_v, and p_z, has exactly one incident curve in S, and (v) no curve in S connects two points of R both lying on s_{uv}, or both lying on s_{vz}, or both lying on s_{zu}. See Fig. 4(a). We show the following.

Lemma 5. *There exists a planar drawing Γ_2 of G_2 such that:*

(a) Vertices u_2, v_2, and z_2 are mapped to p_u, p_v, and p_z, respectively;

(b) edges (u_2, v_2), (v_2, z_2), and (z_2, u_2) are represented by curves s_{uv}, s_{vz}, and s_{zu}, respectively;

(c) the internal vertices of G_2 are mapped to the points of P;

(d) each edge of G_2 that connects two points $p_1, p_2 \in P \cup \{p_u, p_v, p_z\}$ such that there exists a curve $s \in S$ connecting p_1 and p_2 is represented by s in Γ_2; and

(e) each edge e of G_2 and each curve $s \in S$ such that e is not represented by s in Γ_2 cross at most once.

Proof sketch: The statement is proved by induction on n. If $n = 3$, then drawing Γ_2 is easily constructed by mapping vertices u_2, v_2, and z_2 to p_u, p_v, and p_z, respectively, and by mapping edges (u_2, v_2), (v_2, z_2), and (z_2, u_2) to curves s_{uv}, s_{vz}, and s_{zu}. Suppose that $n > 3$. By the properties of plane 3-trees, G_2 has an internal vertex w_2 that is connected to all of u_2, v_2, and z_2. Also, the subgraphs G_2^{uv}, G_2^{vz}, and G_2^{zu} of G_2 induced by the vertices inside or on the border of cycles $C_2^{uv} = (u_2, v_2, w_2)$, $C_2^{vz} = (v_2, z_2, w_2)$, and $C_2^{zu} = (z_2, u_2, w_2)$, respectively, are plane 3-trees with n_{uv}, n_{vz}, and n_{zu} internal vertices, respectively, where $n_{uv} + n_{vz} + n_{zu} = n - 4$. Then, the proof of the lemma is completed by proving that there exists a point $p_w \in P$ and three curves s_{uw}, s_{vw}, and s_{zw} connecting p_u and p_w, connecting p_v and p_w, and connecting p_z and p_w, respectively, such that s_{uw}, s_{vw}, and s_{zw} split $cl(\Delta)$ into three regions $cl(\Delta_{uvw})$, $cl(\Delta_{vzw})$, and $cl(\Delta_{zuw})$ satisfying the same properties $cl(\Delta)$ satisfies. Hence, induction can be applied three times, to construct a drawing Γ_2^{uv} of G_2^{uv} in $cl(\Delta_{uvw})$, to construct a drawing Γ_2^{vz} of G_2^{vz} in $cl(\Delta_{vzw})$, and to construct a drawing Γ_2^{zu} of G_2^{zu} in $cl(\Delta_{zuw})$. Placing Γ_2^{uv}, Γ_2^{vz}, and Γ_2^{zu} together results in a drawing Γ_2 of G_2 satisfying Properties (a)–(e) of the lemma. □

Fig. 4(b) shows a planar drawing of G_2 satisfying the properties of Lemma 5. Lemma 5 implies a proof of Theorem 3. Namely, construct any planar drawing Γ_1 of G_1. Denote by P the point set to which the $n - 3$ internal vertices of G_1 are mapped in Γ_1. Let s_{uv}, s_{vz}, and s_{zu} be the curves representing edges (u_1, v_1), (v_1, z_1), and (z_1, u_1) in Γ_1, respectively. Let S be the set of curves representing the internal edges of G_1 in Γ_1. Let p_u, p_v, and p_z be the points on which u_1, v_1, and z_1 are drawn, respectively. Let $R = \{p_u, p_v, p_z\}$. Construct a planar drawing Γ_2 of G_2 satisfying the properties of Lemma 5. Then, Γ_1 and Γ_2 are planar drawings of G_1 and G_2, respectively. By Properties (a) and (c) of Lemma 5, the n vertices of G_2 are mapped to the same n points to which the vertices of G_1 are mapped. Finally, by Properties (b) and (d) of Lemma 5, if edges e_1 of G_1 and e_2 of G_2 have their end-vertices mapped to the same two points $p_a, p_b \in P \cup \{p_u, p_v, p_z\}$, then e_1 and e_2 are represented by the same Jordan curve in Γ_1 and in Γ_2; hence, Γ_1 and Γ_2 are a SEFENOMAP of G_1 and G_2.

4 Conclusions

In this paper we studied the problem of determining the largest $k_1 \leq n$ such that every n-vertex planar graph and every k_1-vertex planar graph admit a SEFENOMAP . We proved that $k_1 \geq n/2$. No upper bound smaller than n is known. Hence, tightening this bound (and in particular proving whether $k_1 = n$ or not) is a natural research direction.

To achieve the above result, we proved that every n-vertex plane graph has an $(n/2)$-vertex induced outerplane graph, a result related to a famous conjecture stating that every planar graph contains an induced forest with half of its vertices [2]. A suitable triangulation of a set of nested 4-cycles shows that $n/2$ is a tight bound for our algorithm, up to a constant. However, we have no example of an n-vertex plane graph whose largest induced outerplane graph has less than $2n/3$ vertices (a triangulation of a set of nested 3-cycles shows that the $2n/3$ bound cannot be improved). The following question arises: What are the largest k_2 and k_3 such that every n-vertex plane graph has an induced outerplane graph with k_2 vertices and an induced outerplanar graph with k_3 vertices? Any bound $k_2 > n/2$ would improve our bound for the SEFENOMAP problem, while any bound $k_3 > 3n/5$ would improve the best known bound for Conjecture 1, via the results in [9].

A different technique to prove that every n-vertex planar graph and every k_4-vertex planar graph have a SEFENOMAP is to ensure that a mapping between their vertex sets exists that generates no shared edge. Thus, we ask: What is the largest $k_4 \leq n$ such that an injective mapping exists from the vertex set of any k_4-vertex planar graph and the vertex set of any n-vertex planar graph generating no shared edge? It is easy to see that $k_4 \geq n/4$ (a consequence of the four color theorem [11,12]) and that $k_4 \leq n - 5$ (an n-vertex planar graph with minimum degree 5 does not admit such a mapping with an $(n - 4)$-vertex planar graph having a vertex of degree $n - 5$).

Finally, it would be interesting to study the geometric version of our problem. That is: What is the largest $k_5 \leq n$ such that every n-vertex planar graph and every k_5-vertex planar graph admit a geometric simultaneous embedding with no mapping? Surprisingly, we are not aware of any super-constant lower bound for the value of k_5.

Acknowledgments. Research partially supported by NSERC Canada, by the Australian Research Council FT100100755, and by ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations".

References

1. Akiyama, J., Watanabe, M.: Maximum induced forests of planar graphs. Graphs and Combinatorics 3(1), 201–202 (1987)
2. Albertson, M.O., Berman, D.M.: A conjecture on planar graphs. In: Bondy, J.A., Murty, U.S.R. (eds.) Graph Theory and Related Topics, p. 357. Academic Press (1979)
3. Angelini, P., Evans, W., Frati, F., Gudmundsson, J.: SEFE with no mapping via large induced outerplane graphs in plane graphs. CoRR, abs/1309.4713 (2013)
4. Blasiüs, T., Kobourov, S.G., Rutter, I.: Simultaneous embedding of planar graphs. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization. CRC Press (2013)
5. Borodin, O.V.: On acyclic colourings of planar graphs. Disc. Math. 25, 211–236 (1979)
6. Bose, P.: On embedding an outer-planar graph on a point set. Computational Geometry: Theory and Applications 23, 303–312 (2002)
7. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. 36(2), 117–130 (2007)
8. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified points. Amer. Math. Monthly 98(2), 165–166 (1991)
9. Hosono, K.: Induced forests in trees and outerplanar graphs. Proc. Fac. Sci. Tokai Univ. 25, 27–29 (1990)
10. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
11. Kenneth, A., Haken, W.: Every planar map is four colorable part I. discharging. Illinois J. Math. 21, 429–490 (1977)
12. Kenneth, A., Haken, W., Koch, J.: Every planar map is four colorable part II. reducibility. Illinois J. Math. 21, 491–567 (1977)
13. Kratochvíl, J., Vaner, M.: A note on planar partial 3-trees. CoRR, abs/1210.8113 (2012)
14. Pelsmajer, M.J.: Maximum induced linear forests in outerplanar graphs. Graphs and Combinatorics 20(1), 121–129 (2004)
15. Poh, K.S.: On the linear vertex-arboricity of a planar graph. Journal of Graph Theory 14(1), 73–75 (1990)

Hardness and Algorithms for Variants of Line Graphs of Directed Graphs

Mourad Baïou[1], Laurent Beaudou[1,⋆], Zhentao Li[2], and Vincent Limouzy[1,⋆⋆]

[1] Limos, Cnrs and Univ. Clermont II
[2] Lip, Cnrs and ENS Lyon

Abstract. Given a directed graph $D = (V, A)$ we define its intersection graph $I(D) = (A, E)$ to be the graph having A as a node-set and two nodes of $I(D)$ are adjacent if their corresponding arcs share a common node that is the tail of at least one of these arcs. We call them facility location graphs since they arise from the classical uncapacitated facility location problem. In this paper we show that facility location graphs are hard to recognize but they are easy to recognize when the underlying graph is triangle-free. We also determine the complexity of the vertex coloring, the stable set and the facility location problem for triangle-free facility location graphs.

1 Introduction

In this paper we study the following class of intersection graphs. Given a directed graph $D = (V, A)$, we denote by $I(D) = (A, E)$ the *intersection graph of* D defined as follows:

- the node-set of $I(D)$ is the arc-set of D,
- two nodes $a = (u, v)$ and $b = (w, t)$ of $I(D)$ are adjacent if one of the following holds: (1) $u = w$, (2) $v = w$, (3) $t = u$, (4) $(u, v) = (t, w)$ (see Figure 1(a)).

We focus on two aspects: the recognition of these intersection graphs and some combinatorial optimization problem in this class. De Simone and Mannino [1] considered the recognition problem and provided a characterization of these graphs based on the structure of the (directed) neighbourhood of a vertex. Unfortunately this characterization does not yield a polynomial time recognition algorithm.

Intersection graphs we consider arise from the *uncapacitated facility location problem* (UFLP) defined as follows. We are given a directed graph $D = (V, A)$, costs $f(v)$ of opening a facility at node v and cost $c(u, v)$ of assigning v to u (for each $(u, v) \in A$). We wish to select a subset of facilities to open and an assignment of each remaining node to a selected facility so as to minimize the cost of opening the selected facilities plus the cost of arcs used for assignments.

⋆ Supported by Région Auvergne project M2F.
⋆⋆ Supported by the ANR junior project DORSO.

This problem can be formulated as a linear integer program equivalent to maximal clique formulation of the *maximum stable set problem* associated with $I(D)$, where the weight of each node (u, v) of $I(D)$ is $f(u) - c(u, v)$. This correspondence is well known in the literature (see [2,3,1]). We may consider several combinatorial optimization problems on directed graphs that may be reduced to the maximum stable set problem on an undirected graph. For example, Chvátal and Ebenegger [4] reduce the max cut problem in a directed graph $D = (V, A)$ to the maximum stable set problem in the following intersection graph called the *line graph of a directed graph*: we assign a node to each arc $a \in A$ and two nodes are adjacent if the head of one (corresponding) arc is the tail of the other. They prove that recognizing such graphs is NP-complete. Balas [5] considered the asymmetric assignment problem. He defined an intersection graph of a directed graph D where nodes are arcs of D and two nodes are adjacent if the two corresponding arcs have the same tail, the same head or the same extremities without being parallel. Balas uses this correspondence to develop new facets for the asymmetric assignment polytope.

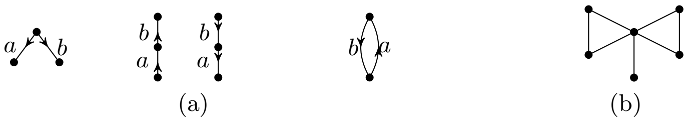


Fig. 1. (a)The adjacency of two nodes a and b in $I(D)$ (b) A graph which is not a FL graph

We may generalize the notion of line graphs to directed graphs in many ways. The simplest involves deciding

1. if arcs that share a head are adjacent,
2. if arcs that share a tail are adjacent, and
3. if two arcs are adjacent when the head of one arc is the tail of the other.

It is not too difficult to show the recognition problem is easy if we choose non-adjacency for (3).

So suppose arcs of type (3) are adjacent. Choosing adjacency for (1) and (2) gives the line graphs of the underlying undirected graph, and these are easy to recognize [6]. Choosing non-adjacency for both (1) and (2) leads to the line graphs defined by Chvátal and Ebenegger and it is NP-complete to recognize them [4]. And picking exactly one of (1) and (2) to be adjacent and non-adjacency for the other leads to the same class of graphs (as we can simply reverse all arcs of a digraph before taking its line graph) and we wish to determine the complexity of recognizing this very last class.

Finally, note that since the stable set problem in our class is equivalent to the facility location problem, one may use tools developed for facility location problem to solve the stable set problem on these graphs. It is well known that in practice the facility location problem may be solved efficiently via several approaches: polyhedra, approximation algorithms and heuristics.

This paper is organized as follows. Section 2 contains some basic definitions and notations. Other definitions and notations will be given when needed. In Section 3 we show that the subclass of triangle-free facility location graphs are recognizable in linear time and, in contrast, in Section 4, we show that facility location graphs are hard to recognize. Section 5 is devoted to some combinatorial optimization problem in facility location graphs. In particular we show that the maximum stable set problem remains NP-complete in triangle-free facility location graphs but the vertex coloring problem is solvable in polynomial time in this class. We also discuss the facility location problem and show it is NP-complete in some restricted class of graphs. We omit some of the more routine but tedious proofs in this extended abstract but make available a complete preprint [7].

2 Definitions and Notations

Let G be an undirected graph, we say that G is a *facility location* (FL) graph if there exists a directed graph D such that $G = I(D)$. D is the *preimage* of G.

Let $D = (V, A)$ be a directed graph. For an arc $a = (u, v) \in A$, we say $t(a) = u$ is the *tail* of a and $h(a) = v$ is the *head* of a. A *sink* is a node which is a tail of no arc in A. A *branch* is an arc a where $h(a)$ is not the head or tail of any other arc.

An undirected graph G is triangle-free if it does not contain a clique of size 3. A *wheel* W_n is a graph obtained from a cycle C_n by adding a vertex adjacent to all vertices of the cycle.

3 Recognizing Triangle-Free Facility Location Graphs

In Section 4, we will show recognizing FL graphs is NP-complete. Our reduction constructs a graph with many cliques of size 3 but none of size 4. Hence it is natural to ask if the recognition problem remains difficult for triangle-free FL graphs. In this section, we show triangle-free FL graphs can be recognized in polynomial time.

In subsection 3.1 we examine the structure of general FL graphs. In subsection 3.2, we prove the main result of this section.

3.1 Structural Properties of Facility Location Graphs

Here, we state some basic properties of FL graphs.

Remark 1. *The preimage of any cycle either contains two arcs with the same tail or is a directed cycle.*

We will use the next three propositions as reduction rules in a recognition algorithm. Each rule allows us to find some structure in the graph and recurse on a simpler graph until no rules apply. If such a simpler graph is also triangle-free then it has a very specific form that is easy to recognize.

Remark 2. *If u is a degree one vertex adjacent to a degree two vertex in an undirected graph G then G is a FL graph if and only if $G - u$ is a FL graph.*

Lemma 1. *If G is a FL graph, then there exists a digraph D such that $G = I(D)$ and every sink node in D has exactly one entering arc.*

Lemma 2. *If u and v are adjacent degree two vertices in G with no common neighbours in an undirected graph G then G is a FL graph if and only if $G - uv$ is a FL graph.*

3.2 Application to Triangle-Free Facility Location Graphs

We now characterizes FL graphs on which no rules from the previous section apply. The *outdegree* of a vertex is the number of arcs leaving that vertex.

Lemma 3. *Let G be a connected triangle-free graph with no degree two vertex adjacent to degree ≤ 2 vertices. If $G = I(D)$ then vertices of D have outdegree at most two. Furthermore, vertices with outdegree exactly 2 have a sink as one of their outneighbours.*

Theorem 3. *Let G be a triangle-free graph. Let G' be the graph obtained from G by removing all edges between degree two vertices. Then G is a FL graph if and only if G' has at most one cycle per connected component.*

Proof. By Lemma 2 (and since G is triangle-free so no adjacent vertices share a common neighbour), we only need to show that G' is a FL graph if and only if G' has at most one cycle per component. Note that G' is also triangle-free.

Necessity. Let G' be a triangle-free facility location graph. By Lemma 1, there is a D where every sink has indegree one with $G' = I(D)$. By Remark 1 and Lemma 3, the preimage of every cycle in G' is a directed cycle.

Suppose that a connected component of G' contains two cycles C_1 and C_2. Since both their preimages are directed cycles, the preimages of any vertex in both C_1 and C_2 is a common arc in both directed cycles and this leads to a triangle in G' (by taking a common arc and two differing arcs following it, one in each cycle). Thus, C_1 and C_2 are vertex disjoint. Since C_1 and C_2 belong to the same connected component, there is a path P in G' from some $u \in C_1$ to $v \in C_2$. Then the image of the second vertex of P points towards the image of C_1 and the image of the second to last vertex of P points towards the image of C_2. So the image of P is not a directed path. But then some two consecutive vertices in P have an image that share a common tail, a contradiction to Lemma 3.

Sufficiency. Consider a connected component of G. Suppose that it consists of a tree. Let us construct a directed graph D with $G = I(D)$. Pick any node r as a root. Let $r = (u_0, v_0)$. Let $r_1, \ldots, r_k$ be the children of r in G, we set $r_i = (v_i, u_0)$ for $i = 1, \ldots, k$. Now each node r_i play the role of r and we repeat this step. This procedure ends with a directed graph D such that $G = I(D)$.

Suppose that there is a cycle C. This cycle must be chordless. Let C' be a directed cycle where each arc in C' correspond to a node in C. The rest of this component consist of disjoint trees each intersect C in one node. If this node is chosen to be the root of the tree, then the procedure above may be applied to get a directed graph D such that $G = I(D)$. □

We are now ready to describe our recognition algorithm, which is the main result of this section.

Theorem 4. *Given an undirected triangle-free graph $G = (V, E)$, we may decide whether or not G is a facility location graph in $O(|E|)$.*

Proof. In $O(|E|)$ we may remove all the edges $e = bc$ with both b and c of degree two. Then we apply a breadth-first search in $O(|E|)$. If a node is encountered more than twice or there are two nodes that were encountered twice, then there are two cycles. Otherwise G is a facility location graph. □

4 Recognizing Facility Location Graphs is NP-complete

The main result of this section is the following theorem.

Theorem 5. *Recognizing facility location graphs is NP-complete.*

We will reduce the problem 3-SAT to the recognition of FL graphs. We assume we are given an instance of 3-SAT. That is, variables $x_1, \dots, x_n$ and a Boolean formula $F = C_1 \wedge \cdots \wedge C_m$, where each clause $C_j = \lambda_{j_1} \vee \lambda_{j_2} \vee \lambda_{j_3}$, for $j = 1 \dots, m$. We construct an undirected graph G_F from F and we show that F is satisfiable if and only if G_F is a facility location graph.

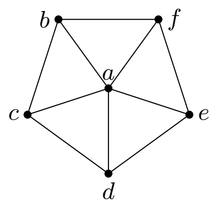
We build G_F using gadgets for variables and clauses. Values for variables are stored, replicated and negated through the "branches" of the variable gadgets. These branches are then connected to the clauses gadgets of clauses that contain these variables (and their negation).

More precisely, the construction of G_F follows three steps: (1) for each variable x_i, we construct a graph called GAD_i^1 (GAD stands for gadget), (2) for each clause C_j, another gadget called GAD_j^2 is constructed and (3) we connect the graphs GAD_i^1 and GAD_j^2 to produce G_F. Each graph GAD_i^1 contains $2m$ branches where each branch express the fact that the variable x_i (or $\bar{x}_i$) is present in the clause C_j, $j = 1, \dots, m$. Each graph GAD_j^2 contains exactly three branches where each branch expresses the literals of this clause λ_{j_1}, λ_{j_2} and λ_{j_3}.

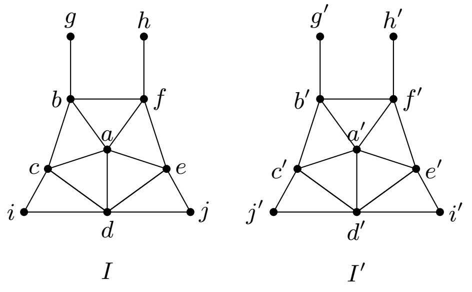
The three following subsections are devoted to the construction of the graphs GAD_i^1, GAD_j^2 and G_F.

4.1 Variable Gadgets

Our variable gadgets is built by identifying vertices in many copies of a graph I with only two preimages. We think of one preimage of I as the Boolean value



(a) The wheel W_5



(b) I and I' are extensions of W_5

Fig. 2.

"true" and the other as "false". I (see Figure 2(b)) is constructed from the wheel W_5 (see Figure 2(a)) by adding four vertices.

We make m copies of I for each variable and combine these copies so that the associated Boolean values are all equal. To ease our analysis, we first examine two copies of I identified on the vertex j in both copies (see Figure 3). Then the preimage of the two copies of I are forced to be the same. We call this graph Inv, the *inverter*, and will use in later construction.

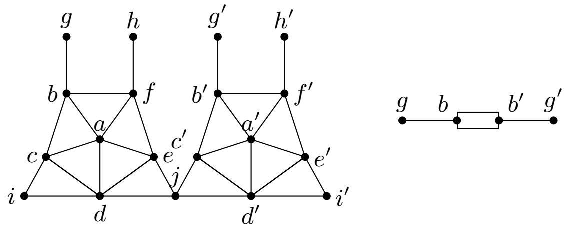


Fig. 3. The graph Inv and its abbreviation

Now if we take m copies of I and identify the node labelled j in a copy with the node labelled i in the next copy (for the first $m-1$ copies, which have successor) then the resulting gadget (see Figure 4) forces all copies of I to have the same preimage. In fact, the preimages of vertices labelled i, j and d in all copies for a directed path.

Lemma 4. *For each directed graph D with $I(D) = \text{Gad}_i^1$ exactly one of the following two assumptions holds:*

(i) $h(b_j^i) = t(g_j^i)$ *and* $t(f_j^i) = h(h_j^i)$ *for each* $j = 1, \ldots, m$,
(ii) $t(b_j^i) = h(g_j^i)$ *and* $h(f_j^i) = t(h_j^i)$ *for each* $j = 1, \ldots, m$.

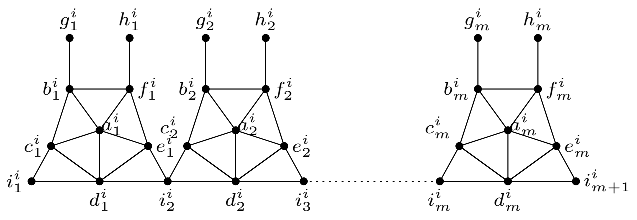
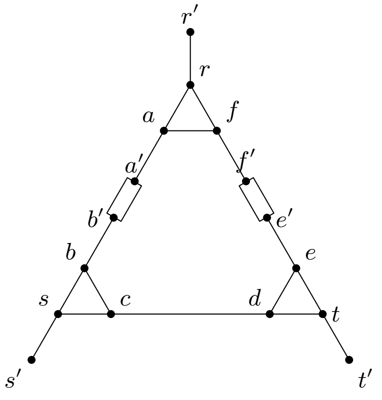


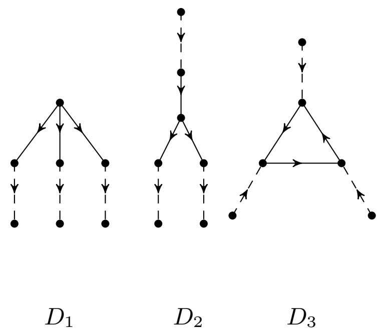
Fig. 4. Graph for every variable x_i, GAD_i^1

4.2 Clause Gadgets

We now describe how to build clause gadgets (see Figure 5(a) for one such gadget and the labelling we use). They are built from three nets (triangles with three degree 1 vertices, one adjacent to each vertex of the triangle) and two copies of INV (of the previous section) by identifying some edges.



(a) Graph Gad_j^2 for clause C_j



(b) The solid line are the nodes of the triangle Δ and the dashed lines are its pending nodes

Fig. 5.

We begin with the following remark.

Remark 6. *There are only three possible preimages for a triangle of a net: those shown in Figure 5(b).*

Lemma 5. *No preimage of* GAD_j^2 *has* $h(r'_j) = t(r_j)$, $h(s'_j) = t(s_j)$ *and* $h(t'_j) = t(t_j)$.

However, if we pick any proper subset of the above three contraints then there is a preimage of GAD_j^2 *where those constraints are satisfied and no other constraints are satisfied.*

4.3 Reduction from 3-SAT

As discussed at the beginning of this section, we build a graph G_F from a Boolean formula F. Build a copy of GAD_i^1 for each Boolean variable x_i and a copy of GAD_j^2 for each clause C_j. We combine some of these disjoint gadgets as follows.

We connect GAD_i^1 and GAD_j^2 through their branches if x_i or $\bar{x}_i$ appears in C_j. Specifically, for each clause $C_j = \lambda_{j_1} \vee \lambda_{j_2} \vee \lambda_{j_3}$ we identify the following vertices:

if $\lambda_{j_1} = x_{j_1}$, we identify r_j with $g_j^{j_1}$ and r'_j with $b_j^{j_1}$,

if $\lambda_{j_1} = \bar{x}_{j_1}$, we identify r_j with $h_j^{j_1}$ and r'_j with $f_j^{j_1}$.

We proceed in the same way for remaining literals λ_{j_2} and λ_{j_3}.

4.4 Proof of Theorem 5

Since the problem 3-SAT is NP-complete, it is sufficient to prove that a Boolean formula F is true if and only if the graph G_F we build is a facility location graph.

Suppose G_F is a facility location graph with preimage D. Then we claim F evaluates to true with the following assignment.

$$x_i = \begin{cases} \text{true} & \text{if the arc } g_1^i \text{ enters the arc } b_1^i \text{ in } D, \\ \text{false} & \text{otherwise} \end{cases}$$

We say an arc a *enters* an arc b if $h(a) = t(b)$.

Notice that from Lemma 4 whenever the arc g_1^i enters the arc b_1^i, then g_j^i enters the arc b_j^i for each $j = 1, \dots, m$. Let C_j be any clause of F. From Lemma 5 (i), we must have that r_j enters r'_j, or s_j enters s'_j or that t_j enters t'_j in any directed graph whose intersection graph is GAD_j^2. We may assume that r_j enters r'_j. By the definition of G_F the branch $r_j r'_j$ is identified with $g_j^i b_j^i$ when x_i is present in C_j and in this case $x_i = 1$ and so $C_j = 1$. Otherwise the branch $r_j r'_j$ is identified with $h_j^i f_j^i$ when $\bar{x}_i$ is present in C_j. So the arc h_j^i enters the arc f_j^i and from Lemma 4 we have that the arc b_j^i enters the arc g_j^i and by definition we have $x_i = 0$, which implies that $C_j = 1$.

Now assume that there is an assignment of the variables x_i, $i = 1, \dots, n$ for which F evaluates to true. We build a preimage of G_F as follows. By Lemma 4, we can (independently) build preimages for each GAD_i^1 so that g_j^i enters b_j^i if x_i is true and b_j^i enters g_j^i if x_i is false. Now given a clause $C_j = \lambda_{j_1} \vee \lambda_{j_2} \vee \lambda_{j_3}$, from Lemma 5 there is no preimage only when the assumption (i) of Lemma 5 is not satisfied. But one can check that this may happen only when all of $\lambda_{j_1}, \lambda_{j_2}, \lambda_{j_3}$ are false, which is not possible.

5 Consequences and Related Problems

5.1 The Vertex Coloring Problem

A *vertex coloring* of a graph is an assignment of colors to the nodes of the graph such that no two adjacent nodes receive the same color. The minimum number needed for a such coloring is called the *chromatic number* and denoted by $\chi(G)$. It is well known that finding $\chi(G)$ is NP-complete for triangle-free graphs. A direct consequence of the previous section shows that when G is a triangle-free facility location graph, it is 2-degenerate, and therefore $\chi(G) \leq 3$.

Theorem 7. *If G is a triangle-free facility location graph, then $\chi(G) \leq 3$. Moreover, $\chi(G)$ may be computed in $O(|E|)$.*

A natural question arises: whether or not coloring facility locations graphs is polynomial. Unfortunately this problem is NP-complete by a reduction from the edge coloring problem (i.e., the vertex coloring problem for line graphs).

Theorem 8. *Coloring facility locations graphs is* NP*-complete.*

Proof. Given an input graph G that we wish to k edge color (a task that is NP-complete [8]), we build an auxiliary digraph D obtained from vertices $V(G)$ with no arcs by adding a vertex x_{uv} for each edge $uv \in E(G)$ with entering arcs from u and v and $k-1$ leaving arcs to $k-1$ new vertices. Now any k vertex coloring of $I(D)$ forces vertices corresponding to both entering arcs of x_{uv} to be colored the same as vertices of leaving arcs from x_{uv} take up $k-1$ colors. It is easy to see that a k vertex coloring of $I(D)$ leads to a k edge coloring of G by giving $e \in E$ the color of arcs entering x_e. Similarly, a k edge coloring of G gives a k vertex coloring of $I(D)$. $\square$

5.2 The Stable Set Problem

Given an undirected graph $G = (V, E)$, a subset of nodes $S \subseteq V$ of an undirected graph is called a *stable set* if there is no edge between any two nodes of S. The *maximum stable set problem* is to find a stable set of maximum size. This size is usually called the *stability number* and denoted by $\alpha(G)$. If we associate a weight $w(v)$ to each vertex $v \in V$, then the *maximum weighted stable set problem* is to find a stable set S with $\sum_{v \in S} w(v)$ maximum.

The maximum stable set problem is NP-complete for triangle-free graph. Poljak [9] showed this by building an auxiliary graph Sub_G from an input graph $G = (V, E)$ by replacing any edge $e = uv$ in E by a path $uu', u'u'', u''v$. Now Sub_G is triangle-free and $\alpha(\text{Sub}_G) = \alpha(G) + |E|$. By Theorem 3, Sub_G is also a facility location graph since the removal of the edges $u'u''$ yields a graph where each connected component is a star. Hence, we obtain the following result,

Theorem 9. *The maximum stable set problem is* NP*-complete in triangle-free facility location graphs.*

Since any triangle-free facility location graph can be colored with 3 colors in $O(|E|)$ Theorem 7, we can get a 3-approximation algorithm for the maximum (weighted) stable set problem. Indeed, we may assume the input graph G has only positive weights and pick the color class S of maximum (total) weight. Now S has weight at least a third of the weight of all of G which is at least a third of the weight of largest stable set in G.

5.3 The Facility Location Problem

Recall that the uncapacitated facility location problem (UFLP) associated with a directed graph D is equivalent to the maximum weighted stable set problem for $I(D)$. Therefore, from Theorem 9 we have the following corollary.

Corollary 10. *The uncapacitated facility location problem remains* NP*-complete even when the input digraph does not contain the four graphs of Figure 6(a) as subgraphs.*

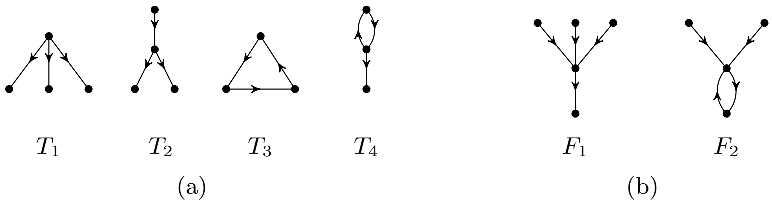


Fig. 6. (a)Graphs T_1, T_2, T_3 and T_4 (b)Graphs F_1 and F_2

Mohar proved the following result in [10].

Theorem 11. *[10] The maximum stable set problem in 2-connected cubic planar graphs is* NP*-complete.*

This results allows us to strengthen Corollary 10 to the following theorem.

Theorem 12. *The uncapacitated facility location problem is* NP*-complete for graphs that do not contain any of T_1, T_2, T_3, T_4, F_1 and F_2 as a subgraph.*

Proof. Let $G = (V, E)$ be an undirected 2-connected cubic planar graph. From G define the subdivision of it, $\textsc{Sub}_G$, as in the previous subsection, that is each edge $e = uv \in E$ is replaced by path of size three. Now we construct a directed graph D containing none of the graphs T_1, T_2, T_3, T_4, F_1 and F_2 as a subgraph and such that $I(D) = \textsc{Sub}_G$. Thus from Theorem 11 the maximum weighted stable set problem is NP-complete in 2-connected cubic planar graphs, and by equivalence we have that UFLP is also NP-complete in graphs satisfying the theorem's hypothesis. Now let us give the construction of D.

From Petersen's theorem the graph G contains a perfect matching M. Let G' be the graph obtained by removing M. Each component of G' is a chordless cycle. Let $C = v_0, v_1, \ldots, v_p$ be one of these cycles. In $\textsc{Sub}_G$ this cycle corresponds to a cycle $C' = v_0, v_1, v_2, \ldots, v_{3p}, v_{3p+1}, v_{3p+2}$. Let us construct a directed graph D with $I(D) = \textsc{Sub}_G$. Each cycle C' of $\textsc{Sub}_G$ may be defined in D by the directed cycle where the arc v_i enters the arc v_{i+1} for each $i = 0, \ldots, 3p + 1$, and the arc v_{3p+2} enters the arc v_0 (an arc a enters an arc b means that the head of a coincide with the tail of b). To complete the definition of D we need to consider all the edges of M and their subdivisions. Let $e = uv \in M$ and u_1, u_2, u_3, u_4 the corresponding path in $\textsc{Sub}_G$. Complete the construction of D by creating for every such edge e two arcs u_2 and u_3 having the same tail where u_2 enters the arc u_1 and u_3 enters the arc u_4.

References

1. De Simone, C., Mannino, C.: Easy instances of the plant location problem. Technical Report R. 427, IASI, CNR (1996)
2. Avella, P., Sassano, A.: On the p-median polytope. Mathematical Programming 89, 395–411 (2001)
3. Cornuejols, G., Thizy, J.M.: Some facets of the simple plant location polytope. Math. Program. 23, 50–74 (1982)
4. Chvátal, V., Ebenegger, C.: A note on line digraphs and the directed max-cut problem. Discrete Applied Mathematics 29, 165–170 (1990)
5. Balas, E.: The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. SIAM Journal on Discrete Mathematics 2, 425–451 (1989)
6. Beineke, L.W.: Characterizations of derived graphs. Journal of Combinatorial Theory 9, 129–135 (1970)
7. Baïou, M., Beaudou, L., Li, Z., Limouzy, V.: On a class of intersection graphs (2013), http://arxiv.org/abs/1306.2498
8. Holyer, I.: The NP-completeness of edge-coloring. SIAM Journal on Computing 10, 718–720 (1981)
9. Poljak, S.: A note on stable sets and colorings of graphs. Commentationes Mathematicae Universitatis Carolinae 15, 307–309 (1974)
10. Mohar, B.: Face covers and the genus problem for apex graphs. J. Comb. Theory, Ser. B 82, 102–117 (2001)

Performance Guarantees for Scheduling Algorithms under Perturbed Machine Speeds [⋆]

Michael Etscheid

Dept. of Computer Science, University of Bonn, Germany
etscheid@cs.uni-bonn.de

Abstract. We study two local search and a greedy algorithm for scheduling. The worst-case performance guarantees are well-known but seem to be contrived and too pessimistic for practical applications. For unrestricted machines, Brunsch et al. [3] showed that the worst-case performance guarantees of these algorithms are not robust if the job sizes are subject to random noise. However, in the case of restricted related machines the worst-case bounds turned out to be robust even in the presence of random noise. We show that if the machine speeds rather than the job sizes are perturbed, also the performance guarantees for restricted machines decrease thus yielding a stronger result.

Keywords: smoothed analysis, scheduling, performance guarantees

1 Introduction

For many simple scheduling algorithms, the worst-case performance guarantees are known up to a constant factor. However, the instances used to construct lower bounds seem to be artificial and not practically relevant if there is some noise on the input. Therefore, we use the framework of smoothed analysis to identify worst-case bounds which are too pessimistic with high probability if the input is perturbed. In this section, we define the scheduling problem, introduce the framework of smoothed analysis shortly and compare briefly our results with the worst-case bounds and the bounds given in [3]. In Section 2 and Section 3, we give some ideas for our proofs for the settings of unrestricted and restricted machines, respectively. Many formal proofs and technical details are omitted due to space constraints. They will be included in a full version of this paper.

The Scheduling Problem. Let $J = \{1, \dots, n\}$ be the set of jobs and $M = \{1, \dots, m\}$ be the set of machines on which the jobs shall be processed. Each machine $i \in M$ has a speed s_i and each job $j \in J$ has a processing requirement p_j. The speeds of the fastest and the slowest machine are denoted by $s_{\max}$ and $s_{\min}$, respectively. We consider two different environments: In the case of unrestricted machines, each job is allowed to run on any machine. In the case of restricted

[⋆] This research was supported by ERC Starting Grant 306465 (BeyondWorstCase).

machines, each job $j \in J$ has a set $\mathcal{M}_j \subseteq M$ of allowed machines. These variables form an instance I of the scheduling problem.

A function $\sigma : J \to M$ is called a schedule. The time a machine i needs to process job j is p_j/s_i if job j is allowed to run on machine i, and ∞ otherwise. Given a schedule σ for an instance I, the load of a machine i is defined as $L_i(I, \sigma) = \sum_{j \in \sigma^{-1}(i)} p_j/s_i$. The makespan is defined as $C_{\max}(I, \sigma) = \max_{i \in M} L_i(I, \sigma)$. We write $C^*_{\max}(I)$ for an optimal makespan. The goal is to minimize the makespan. Sometimes we omit the parameters I and σ, respectively, if they are clear from the context.

Studied Scheduling Algorithms. We study a greedy and two local search algorithms.

The *list scheduling* algorithm starts with an empty schedule. Then it iteratively assigns an unscheduled job to the machine on which it will be completed first with respect to the current partial schedule. Any schedule which can be generated this way is called a *list schedule.*

The *jump* and *lex-jump* algorithms start with an arbitrary schedule and then perform local improvement steps. In each step, a job is reassigned to a different machine where it finishes earlier. We assume here that all jobs assigned to a machine finish at the same time, which is the load of the machine. In the jump algorithm, only jobs assigned to a *critical* machine, i.e., a machine with maximal load, are allowed to be reassigned whereas the lex-jump algorithm does not have this limitation. A schedule which cannot be improved by the (lex-)jump algorithm is called *(lex-)jump optimal.*

We write $\mathit{Jump}(I)$ for the set of all jump optimal schedules for a scheduling instance I.

Smoothed Analysis. The framework of smoothed analysis was introduced by Spielman and Teng [13] to explain the good running time of some algorithms in practice despite a bad worst-case running time. We use the more general model suggested by Beier and Vöcking [2]. Let $\phi \geq 1$ be a parameter for the maximum probability density. A *ϕ-smooth* instance $\mathcal{I}$ consists of job sizes $p_1, \ldots, p_n$, subsets $\mathcal{M}_j \subseteq M, j \in J$, in the case of restricted machines, the number m of machines, and density functions $f_i \colon [0, 1] \to [0, \phi]$ for all machines $i \in \{1, \ldots, m\}$. Each machine speed s_i is then chosen according to the density function f_i independently of the other machine speeds. Thus, any ϕ-smooth instance is a distribution over infinitely many scheduling instances. We then look for ϕ-smooth instances for which the *expected* performance ratio is as bad as possible. For example, we can choose for every s_i an interval of length $1/\phi$ from which it is drawn uniformly at random. For $\phi = 1$, this model complies with an average case analysis, whereas for $\phi \to \infty$ the smoothed analysis tends to a worst-case analysis as the machine speeds can be specified within arbitrary precision.

Related Work and Our Results. Minimizing the makespan in a scheduling instance is a well-known strongly NP-hard problem. There is a polynomial

Table 1. Performance guarantees for unrestricted machines

algorithm	worst-case	perturbed job sizes	perturbed speeds
jump	$\Theta(\sqrt{m})$ [4,12]	$\Theta(\phi)$ [3]	$\Theta(\phi)$
lex-jump	$\Theta\left(\min\left\{\frac{\log m}{\log\log m}, \log\frac{s_{\max}}{s_{\min}}\right\}\right)$ [5]	$\Theta(\log\phi)$ [3]	$\Theta(\log\phi)$
list	$\Theta(\log m)$ [1,4]	$\Theta(\log\phi)$ [3]	$\Theta(\log\phi)$

approximation scheme for the unrestricted case by Hochbaum and Shmoys [7] as well as a 2-approximation algorithm for restricted machines by Lenstra et al. [9]. For the algorithms studied in this paper, Table 1 shows an overview of the worst-case and smoothed performance guarantees in the environment of unrestricted machines. We were able to reproduce the same results as Brunsch et al. [3] with perturbed machine speeds instead of perturbed job sizes. Accordingly, we get the same conclusions that the lex-jump algorithm and the list jump algorithm should perform well in practice. An interesting deduction of theirs is that the smoothed price of anarchy for routing games on parallel links is $\Theta(\log\phi)$ as well, as pure Nash equilibria can be seen as local optima according to the lex-jump algorithm. This result carries over to our smoothed model with perturbed link speeds.

Table 2. Performance guarantees for restricted machines. Here, $S = \sum_{i=1}^{m} \frac{s_i}{s_{\min}}$.

algorithm	worst-case	perturbed job sizes	perturbed machine speeds
jump	$\Theta\left(\sqrt{m \cdot \frac{s_{\max}}{s_{\min}}}\right)$ [11]	$\Theta\left(\sqrt{m \cdot \frac{s_{\max}}{s_{\min}}}\right)$ [3]	$\Theta\left(m\sqrt{\phi}\right)$
lex-jump	$\Theta\left(\frac{\log S}{\log\log S}\right)$ [11]	$\Omega\left(\frac{\log m}{\log\log m}\right)$ [3]	$\Theta\left(\min\left\{m, \frac{\log(m\phi)}{\log\log(m\phi)}\right\}\right)$

As Table 2 shows, the worst-case performance guarantees in the environment of restricted machines are robust against random noise on the job sizes. We calculate the expected values of these worst-case bounds to obtain the smoothed bounds in our model. As the expected speed of the slowest machine is in $\Theta(1/(m\phi))$, the bounds change in an intuitive way. On the other hand, we construct classes of smoothed scheduling instances showing that the resulting upper bounds are tight up to a constant factor.

2 Unrestricted Machines

For unrestricted machines, we are able to yield the same results with perturbed machine speeds as Brunsch et al. [3] did with perturbed job sizes. As the main ideas in this section are similar to the proofs by Brunsch et al., we often refer to this paper and omit the formal proofs.

2.1 Bounds for the Jump Algorithm

For the lower bound, we consider a smoothed scheduling instance with one fast and many slow machines as well as one big and many small jobs. Here, fast and big mean roughly 1 and slow and small mean roughly $1/\phi$, respectively. If we assign the big job to the fast machine and each small job to a single slow machine, we get a constant makespan. If we assign the big job to the fastest slow machine, it does not improve its running time by jumping to a different slow machine even if it is empty. If we are now able to assign enough small jobs to the fast machine such that this machine is not critical but the big job does not jump to it, we gain a jump optimal schedule with a makespan of roughly ϕ. This yields the following theorem:

Theorem 1. *There is a class of ϕ-smooth instances $\mathcal{I}$ with unrestricted and related machines such that for each $I \in \mathcal{I}$,*

$$\max_{\sigma \in \mathrm{Jump}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} = \Omega(\phi)\,.$$

For a matching upper bound, we use the observation that the sum of the machine speeds cannot be too small with high probability due to Hoeffding's bound [8]. We can then use the following lemma, which can also be found for example in [12].

Lemma 1. *Let I be a scheduling instance with unrestricted machines and let $s_i \in (0,1]$ for all $i \in \{1,\dots,m\}$. Then for each $\sigma \in \mathrm{Jump}(I)$,*

$$C_{\max}(I,\sigma) \le \left(1 + \frac{m}{\sum_{i=1}^{m} s_i}\right) C^*_{\max}(I)\,.$$

The remaining parts of the proof of the following theorem are analogous to the corresponding proof by Brunsch et al. [3], which is why we omit them here.

Theorem 2. *For any ϕ-smooth instance $\mathcal{I}$ with unrestricted related machines,*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma \in \mathrm{Jump}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right] < 5.1\phi + 2.5 = \mathcal{O}(\phi)\,.$$

2.2 Upper Bounds for List Schedules and Lex-jump Optimal Schedules

Let w.l.o.g. the machine speeds of a scheduling instance I be sorted in descending order. We derive an upper bound for the performance guarantees of so-called *near list schedules*, which include all list schedules and all lex-jump optimal schedules. We only repeat the definition of near list schedules. For further explanations as well as the proofs of some properties (including the relation between near list schedules, list schedules, and lex-jump optimal schedules), we refer to Brunsch et al. [3]. We then use a notation which goes back to Czumaj and Vöcking [5].

Definition 1. *We call a schedule σ a* near list schedule *if we can index the jobs in such a way that for all machines $i \in M, i' \neq i$ and all jobs $j \in \sigma^{-1}(i)$,*

$$L_{i'} + \frac{p_j}{s_{i'}} \geq L_i - \sum_{\ell \in \sigma^{-1}(i) \cap \{1,\dots,j-1\}} \frac{p_\ell}{s_i}. \tag{1}$$

With $NL(I)$ we denote the set of all near list schedules for instance I.

Definition 2. *For a schedule σ, let*

1. $c := \lfloor C_{\max}(\sigma)/C^*_{\max} \rfloor - 1$.
2. $\forall k \in \{0,\dots,c\} \colon H_k := \{1,\dots,i_k\}$, *where* $i_k = \max\{i \in M \colon L_{i'} \geq k \cdot C^*_{\max} \quad \forall i' \leq i\}$.
3. $\forall k \in \{0,\dots,c-1\} \colon R_k := H_k \setminus H_{k+1}, \quad R_c := H_c$.

It is easy to show that $|H_k| \leq m/k$ for each $1 \leq k \leq c$ since $\sum_{i \in H_k} L_i \geq |H_k| \cdot k \cdot C^*_{\max}$ and $\sum_{i \in M} L_i \leq m \cdot C^*_{\max}$. Therefore, $|R_0 \cup R_1| = |M \setminus H_2| \geq m/2$. From Brunsch et al. [3], we know that machine 1 is in R_c and that, for $0 \leq k_2 \leq k_1 \leq c$ and machines $i_1 \in R_{k_1}, i_2 \in R_{k_2}$, the relation $s_{i_1} \geq s_{i_2} \cdot 2^{\lfloor (k_1-k_2)/6 \rfloor}$ holds. Hence, for each machine $i \in R_0 \cup R_1$, we get $1 \geq s_1 \geq s_i \cdot 2^{\lfloor (c-1)/6 \rfloor}$, i.e., $s_i \leq 2^{\lceil (1-c)/6 \rceil} \leq 2^{(1-c)/6+5/6} = 2^{1-c/6}$, where the last inequality holds due to the integrality of c. So at least half of the machines have a speed which is exponentially small in the makespan. Therefore, it is unlikely that the makespan is much greater than $\log \phi$. Formally, we get the following results.

Lemma 2. *For any $\alpha > 0$,*

$$\Pr_{I \sim \mathcal{I}}\left[\max_{\sigma \in \mathrm{NL}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} \geq \alpha\right] \leq \left(\frac{16\phi}{2^{\alpha/6}}\right)^{m/2}.$$

Theorem 3. *For any ϕ-smooth instance $\mathcal{I}$ with unrestricted related machines,*

$$\mathop{\mathbf{E}}_{I \sim \mathcal{I}}\left[\max_{\sigma \in \mathrm{NL}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right] \leq 18 \log_2 \phi + 24 = \mathcal{O}(\log \phi).$$

2.3 Lower Bounds for List Schedules and Lex-jump Optimal Schedules

For the matching lower bound, we use a similar idea like Brunsch et al. [3] do. The intuition behind it can be found in the mentioned paper.

Theorem 4. *There is a class of ϕ-smooth instances $\mathcal{I}$ with unrestricted and related machines such that for each $I \in \mathcal{I}$,*

$$\max_{\sigma \in \mathrm{Lex}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} = \Omega(\log \phi) \quad \textit{and} \quad \max_{\sigma \in \mathrm{List}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} = \Omega(\log \phi).$$

Proof. For a given instance $I \in \mathcal{I}$, we create a list schedule which is lex-jump optimal. It suffices to show the claim for $\phi \geq 4$. For the sake of notational simplicity, we consider scaled probability densities $f_i : (0, 2^r] \to [0, \phi/2^r]$ where $r = \lfloor \log_4 \phi \rfloor$. It is $2^{2r} \leq \phi$, i.e.,

$$\frac{2^r}{\phi} \leq 2^{-r} \leq \frac{1}{2}\,. \tag{2}$$

We define machine classes $M_0, \ldots, M_r$ with $|M_k| = \frac{r!}{k!}$ for each machine class M_k. Each machine speed s_i for a machine $i \in M_k$ is chosen uniformly from $\left(2^k - \frac{2^r}{\phi}, 2^k\right] \subseteq (0, 2^r]$. Note that these intervals are disjoint due to inequality (2), i.e., the machine classes are sorted in ascending order by speed. Let the set of jobs be partitioned into job classes $J_1, \ldots, J_r$, where each class J_ℓ contains $\frac{r!}{(\ell-1)!}$ jobs of the size 2^ℓ. Use Algorithm 1, which is due to Brunsch et al. [3], to obtain a list schedule σ.

Algorithm 1

```
1. for $k = 1$ to $r$ do
2.     for $\ell = r$ down to $k$ do
3.         Schedule $r!/\ell!$ arbitrary jobs of class $J_\ell$ according to list scheduling.
4.     end for
5. end for
```

First construct a schedule σ' with constant makespan: For any $\ell \in \{0, \ldots, r-1\}$, assign exactly one job from $J_{\ell+1}$ to each machine in M_ℓ. The single machine in M_r remains empty. The running time of a machine $i \in M_\ell$ is bounded from above by

$$L_i(\sigma') \leq \frac{2^{\ell+1}}{2^\ell - \frac{2^r}{\phi}} \leq \frac{2^{\ell+1}}{2^\ell - \frac{1}{2}} \leq \frac{2^{\ell+1}}{2^{\ell-1}} = 4\,.$$

Hence, it suffices that σ has a makespan of at least r and that σ is lex-jump optimal. The following two lemmata conclude the proof of the theorem.

Lemma 3. *For any $\ell \in \{1, \ldots, r\}$, any machine $i \in M_l$ is assigned exactly ℓ jobs of job class J_ℓ and no other jobs. The machines in M_0 remain empty.*

Lemma 4. *The generated schedule σ is lex-jump optimal and its makespan is at least r.*

Proof. For any $\ell \in \{0, \ldots, r\}$ and any machine $i \in M_\ell$, it is $\ell \leq L_i < \ell + 2^{\ell-r}$, which can be shown by a short calculation. Especially, the makespan of σ is at least r as M_r is not empty. Let j be a job assigned to machine i. Then $p_j = 2^\ell$. Let $i \neq i' \in M_{\ell'}$. If we assigned job j to machine i', this machine would have a load of

$$L_{i'} + \frac{p_j}{s_{i'}} \geq \ell' + \frac{2^\ell}{2^{\ell'}} = \ell - (\ell - \ell') + 2^{\ell-\ell'} \geq \ell + 1 \geq \ell + 2^{\ell-r} > L_i\,,$$

where the second inequality follows from $2^k - k \geq 1$ for all $k \in \mathbb{N}$. Hence, no job can improve its running time. □

□

3 Restricted Machines

In this section, we provide matching upper and lower bounds for the performance guarantees of the jump optimal and lex-jump optimal schedules which are smaller than the previously known bounds for the worst case and for perturbed job sizes. Due to space constraints, many proofs are omitted.

3.1 Bounds for the Jump Algorithm

The worst-case upper bound for the performance guarantee of the Jump algorithm on restricted machines is

$$\max_{\sigma \in \mathrm{Jump}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} \leq \frac{1}{2} + \sqrt{\frac{1}{4} + (m-1)\frac{s_{\max}}{s_{\min}}} \tag{3}$$

due to Recalde et al. [11]. In our setting of scaled and perturbed machine speeds, the fraction $s_{\max}/s_{\min}$ cannot be too small in expectation. This way we derive an upper bound for the smoothed performance guarantee.

Theorem 5. *For each ϕ-smooth instance $\mathcal{I}$ with restricted related machines*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma \in \mathrm{Jump}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right] \leq 1 + 2m\sqrt{\phi} = \mathcal{O}\left(m\sqrt{\phi}\right) .$$

We now derive a matching lower bound. It suffices to construct a ϕ-smooth instance which has a bad jump optimal schedule with constant probability, which is 1/9 in our case. We can then assume the smallest machine speed to be in a certain interval.

Lemma 5. *Let $m \geq 2$, $s_1,\dots,s_m$ be drawn independently from the interval $[0,1/\phi]$. Then $\mathbf{Pr}\left[s_{\min} \in \left[\frac{1}{m\phi},\frac{2}{m\phi}\right]\right] \geq \frac{1}{9}$.*

Theorem 6. *For every $\phi > 4$ and $m \geq 6$, there is a ϕ-smooth instance $\mathcal{I}$ with m restricted machines and uniform job sizes such that*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma \in Jump(I)} \frac{C_{max}(I,\sigma)}{C^*_{max}(I)}\right] = \Omega\left(m\sqrt{\phi}\right) .$$

Proof. Let $k = \left\lfloor\frac{m}{3}\right\rfloor \geq 2$. The machine speeds $s_1,\dots,s_k$ are uniformly drawn from $[0,1/\phi]$, whereas the machine speeds $s_{k+1},\dots,s_{2k}$ and $s_{2k+1},\dots,s_m$ are drawn from $\left(\frac{1}{\sqrt{\phi}}-\frac{1}{\phi},\frac{1}{\sqrt{\phi}}\right)$ and $\left[\frac{\phi-1}{\phi},1\right]$, respectively. The jobs are partitioned in two classes $J = J_1 \dot\cup J_2$ with $J_1 = \{1,\dots,k\}$ and $|J_2| = k\left\lceil\sqrt{\phi}\right\rceil$. All jobs have size 1. Job $j \in J_1$ is only allowed to run on the machines j and $k+j$ while each job in J_2 is allowed to run on the machines $1,\dots,3k$.

First consider a schedule σ', which assigns each job $j \in J_1$ to machine $k+j$. The jobs in J_2 are distributed evenly over the k machines $2k+1,\dots,3k$. It follows that $C^*_{\max}(I) \le C_{\max}(I,\sigma') \le \max\left\{\left(\frac{1}{\sqrt{\phi}} - \frac{1}{\phi}\right)^{-1}, \frac{k\lceil\sqrt{\phi}\rceil}{k} \cdot \frac{\phi}{\phi-1}\right\} \le 2\sqrt{\phi}$, where the last inequality follows from a simple calculation using $\phi > 4$.

We now construct a bad jump optimal schedule σ: Let i_0 be the slowest machine. Since $\frac{1}{\phi} \le \frac{1}{\sqrt{\phi}} - \frac{1}{\phi}$, we know that $i_0 \le k$. Let $\mathcal{F}$ be the event that $s_{i_0} \notin \left[\frac{1}{k\phi}, \frac{2}{k\phi}\right]$. If $\mathcal{F}$ occurs, set σ to an arbitrary schedule. Otherwise assign job $i_0 \in J_1$ to machine i_0 and $\left\lceil \frac{s_{k+i_0}}{s_{i_0}} - 1\right\rceil$ jobs from J_2 to machine $k+i_0$. Distribute the remaining jobs in the same way as in σ'. This procedure is well-defined, as $\left\lceil \frac{s_{k+i_0}}{s_{i_0}} - 1\right\rceil \le \frac{s_{k+i_0}}{s_{i_0}} \le \frac{1/\sqrt{\phi}}{1/(k\phi)} = k\sqrt{\phi} \le k\left\lceil\sqrt{\phi}\right\rceil = |J_2|$. Note that this way no further jobs are assigned to the machines i_0 and $k+i_0$ and that every other machine has not a greater load than in σ'. Because of

$$L_{k+i_0} = \frac{\left\lceil \frac{s_{k+i_0}}{s_{i_0}} - 1\right\rceil}{s_{k+i_0}} < \frac{\frac{s_{k+i_0}}{s_{i_0}}}{s_{k+i_0}} = \frac{1}{s_{i_0}} = L_{i_0} \le \frac{\left\lceil \frac{s_{k+i_0}}{s_{i_0}}\right\rceil}{s_{k+i_0}} = L_{k+i_0} + \frac{p_{i_0}}{s_{k+i_0}}$$

and $L_{i_0} = \frac{1}{s_{i_0}} \ge \frac{k\phi}{2} \ge \phi > 2\sqrt{\phi} \ge C_{\max}(I,\sigma') \ge L_i$ for all $i \in M \setminus \{i_0, k+i_0\}$, machine i_0 is the only critical machine and σ is jump optimal with $C_{\max}(I,\sigma) = \frac{1}{s_{i_0}} \ge \frac{k\phi}{2}$. Therefore,

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma\in\mathrm{Jump}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right] \ge 1\cdot\mathbf{Pr}[\mathcal{F}] + \frac{k\phi/2}{2\sqrt{\phi}}\cdot\mathbf{Pr}\left[\overline{\mathcal{F}}\right] \ge \frac{k\sqrt{\phi}}{36} = \Omega\left(m\sqrt{\phi}\right),$$

where we used $\mathbf{Pr}\left[\overline{\mathcal{F}}\right] \ge \frac{1}{9}$ due to Lemma 5. □

3.2 Bounds for the Lex-jump Algorithm

In a deterministic setting, there are two tight upper bounds which do not dominate each other.

Lemma 6. *Let I be a scheduling instance with restricted machines. Then*

$$\max_{\sigma\in\mathrm{Lex}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} \le \Gamma^{-1}(S) = \mathcal{O}\left(\frac{\log S}{\log\log S}\right), \text{ where } S = \sum_{i\in M}\frac{s_i}{s_{\min}},$$

and

$$\max_{\sigma\in\mathrm{Lex}(I)} \frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)} = \mathcal{O}(m).$$

For both bounds exist scheduling instances such that the particular bound is tight up to a constant factor.

The first bound is by Recalde et al. [11] and the second by Garing et al. [6]. Again we obtain a smoothed upper bound by computing the expected values of these worst-case bounds.

Theorem 7. *For every ϕ-smooth instance $\mathcal{I}$ with restricted machines,*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma\in\mathrm{Lex}(I)}\frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right]=\mathcal{O}\left(\min\left\{m,\frac{\log m\phi}{\log\log m\phi}\right\}\right).$$

The proof is essentially an application of Jensen's inequality. Now we want to show a matching lower bound.

Theorem 8. *For every $\phi\geq 6$ and $m\geq 216$, there is a ϕ-smooth instance with m restricted machines such that*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma\in\mathrm{Lex}(I)}\frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right]=\Omega\left(\min\left\{m,\frac{\log m\phi}{\log\log m\phi}\right\}\right).$$

To prove Theorem 8, we use a lemma by Lu and Yu [10]:

Lemma 7.

$$\max\left\{\frac{\log x}{\log\log x},\frac{\log y}{\log\log y}\right\}=\Omega\left(\frac{\log xy}{\log\log xy}\right).$$

Due to this lemma and the well-known fact that $\Gamma^{-1}(x)=\Theta(\log x/\log\log x)$, it suffices to show the lower bound

$$\Omega\left(\min\{m,\max\{\Gamma^{-1}(m),\Gamma^{-1}(\phi)\}\}\right)=\Omega\left(\max\{\min\{m,\Gamma^{-1}(\phi)\},\Gamma^{-1}(m)\}\right).$$

We accomplish this by specifying two families of ϕ-smooth instances for every $\phi\geq 6$ and $m\geq 216$, one with an expected performance guarantee of at least $\min\{m,\Gamma^{-1}(\phi)\}$ and one with an expected performance guarantee of at least $\Gamma^{-1}(m)$. For the first of these two bounds, we modify some instances used by Lu and Yu [10]. For the latter bound, we introduce some instances which are distantly related to the instance used by Brunsch et al [3] to show the lower bound $\Gamma^{-1}(m)$ for perturbed job sizes.

Theorem 9. *For every $\phi\geq 6$ and $m\geq 3$, there exists a ϕ-smooth instance with restricted machines such that*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma\in\mathrm{Lex}(I)}\frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right]=\Omega(\min\{m,\Gamma^{-1}(\phi)\}).$$

Theorem 10. *For every $\phi\geq 4$ and $m\geq 216$, there is a ϕ-smooth instance with restricted machines such that*

$$\mathop{\mathbf{E}}_{I\sim\mathcal{I}}\left[\max_{\sigma\in\mathrm{Lex}(I)}\frac{C_{\max}(I,\sigma)}{C^*_{\max}(I)}\right]=\Omega(\Gamma^{-1}(m)).$$

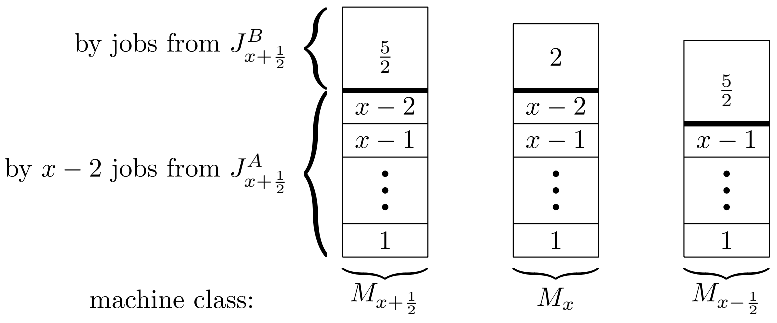


Fig. 1. Idea of the load creation on three machines from different machine classes with average speed, according to the bad schedule σ. Let $x \in \mathbb{N}$. The main load on the second machine is created by $x-2$ big jobs from J^A_x. Many small jobs from J^B_x generate an additional constant load. No big job switches to a neighbored machine as the load differences are too small.

Proof Sketch. Let $r = \max\{3, \lfloor \Gamma^{-1}(\sqrt{m})-4 \rfloor\}$. We construct a ϕ-smooth instance $\mathcal{I}$ with a constant optimum makespan as well as a schedule σ such that σ has a makespan of at least $r - 1/8$ and is lex-jump optimal with constant probability. This yields the theorem.

Let $X = \left\{3, 3+\frac{1}{2}, 4, \ldots, r-\frac{1}{2}, r\right\}$. The set M of machines is partitioned into machine classes M_x of the size $m_x = |M_x|$, $x \in X$. Every machine speed is chosen uniformly from the interval $\left[\frac{\phi-1}{\phi}, 1\right]$. For each machine class M_x there is a job class J^A_x consisting of $\left|J^A_x\right| = \lfloor x-2 \rfloor m_x$ jobs with the size $p^A = \frac{2\phi-1}{2\phi}$, which are only allowed to run on machines in $M_x \cup M_{x-1/2}$, where $M_{2+1/2} = \emptyset$.

If we distribute for each $x \in X$ the jobs from J^A_x to the machines in M_x with the list scheduling algorithm and if we assume that every machine speed is roughly $\frac{2\phi-1}{2\phi}$, then every machine $i \in M_x$ has a load of roughly $\lfloor x-2 \rfloor$. Therefore, if we can show that this schedule is lex-jump optimal, we know that the worst lex-jump optimal schedule has a makespan of at least $r-2$. On the other hand, if we set $m_x = \left|J^A_{x+1/2}\right|$, $x \in X \backslash \{r\}$, we can assign each job in $J^A_{x+1/2}$ to a different machine in M_x yielding a schedule with a constant makespan.

The assumption that the machine speeds are all roughly $\frac{2\phi-1}{2\phi}$ is too optimistic, however. Instead, we can assume due to Hoeffding's bound [8] that the *average* speed of all machines in a machine class M_x is close to $\frac{2\phi-1}{2\phi}$ with high probability. We then introduce new job classes J^B_x for every $x \in X$ with $\left|J^B_x\right| = (2+x-\lfloor x \rfloor)32m_x$ jobs of size $p^B = \frac{p^A}{32}$ which are only allowed to run on machines from M_x. If we assign these small jobs by list scheduling after the big jobs have been distributed, we can assure that the loads of two different machines of one machine class according to the obtained schedule σ do not differ much. Hence, the load of every machine in a machine class M_x is roughly $(\lfloor x-2 \rfloor m_x + (2+x-\lfloor x \rfloor)m_x)/m_x = x$ with an absolute error of at most $1/8$. This construction is depicted in Figure 1.

4 Concluding Remarks

We have shown that a smoothed input model with perturbed machine speeds leads to the same tight bounds for the performance guarantees for unrestricted machines as a model with perturbed job sizes. In particular, these bounds do not depend on the number of machines anymore like the worst-case bounds do. For the setting of restricted machines, we were able to improve the worst-case bounds to their expected values gaining a stronger result than the former work by Brunsch et al. [3]. We conjecture that perturbing both jobs sizes and machine speeds does not result in smaller bounds as the instances used to prove the lower bounds in our setting seem to be robust against random noise on the job sizes. But we heavily used that the sets $\mathcal{M}_j$ are not randomly chosen. We conjecture that the bounds for restricted machines decrease significantly if one is able to justify random noise on the sets $\mathcal{M}_j$ using for example a model from [14].

References

1. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. JACM 44(3), 486–504 (1997)
2. Beier, R., Vöcking, B.: Random knapsack in expected polynomial time. Journal of Computer and System Sciences 69(3), 306–329 (2004)
3. Brunsch, T., Röglin, H., Rutten, C., Vredeveld, T.: Smoothed Performance Guarantees for Local Search. Mathematical Programming (May 2013)
4. Cho, Y., Sahni, S.: Bounds for list schedules on uniform processors. SIAM Journal on Computing 9, 91–103 (1980)
5. Czumaj, A., Vöcking, B.: Tight bounds for worst-case equilibria. ACM Transactions on Algorithms 3(1) (2007)
6. Gairing, M., Lücking, T., Mavronicolas, M., Monien, B.: The Price of Anarchy for Restricted Parallel Links. Parallel Processing Letters 16(1), 117–131 (2006)
7. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. SIAM Journal on Computing 17, 539–551 (1988)
8. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301), 13–30 (1963)
9. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming 46(1-3), 259–271 (1990)
10. Lu, P., Yu, C.: Worst-Case Nash Equilibria in Restricted Routing. Journal of Computer Science and Technology 27(4), 710–717 (2012)
11. Recalde, D., Rutten, C., Schuurman, P., Vredeveld, T.: Local search performance guarantees for restricted related parallel machine scheduling. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 108–119. Springer, Heidelberg (2010)
12. Schuurman, P., Vredeveld, T.: Performance Guarantees of Local Search for Multiprocessor Scheduling. Informs Journal on Computing 19(1), 52–63 (2007)
13. Spielman, D.A., Teng, S.-H.: Smoothed Analysis of Algorithms: Why The Simplex Algorithm Usually Takes Polynomial Time. JACM 51(3), 385–463 (2004)
14. Spielman, D.A., Teng, S.-H.: Smoothed Analysis. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 256–270. Springer, Heidelberg (2003)

Better Bounds for Online k-Frame Throughput Maximization in Network Switches

Jun Kawahara[1,⋆], Koji M. Kobayashi[2,⋆⋆], and Shuichi Miyazaki[3,⋆⋆⋆]

[1] Graduate School of Information Science, Nara Institute of Science and Technology
jkawahara@is.naist.jp

[2] National Institute of Informatics
kobaya@nii.ac.jp

[3] Academic Center for Computing and Media Studies, Kyoto University
shuichi@media.kyoto-u.ac.jp

Abstract. We consider a variant of the online buffer management problem in network switches, called the k-frame throughput maximization problem (k-FTM). This problem models the situation where a large frame is fragmented into k packets and transmitted through the Internet, and the receiver can reconstruct the frame only if he/she accepts all the k packets. Kesselman et al. introduced this problem and showed that its competitive ratio is unbounded even when $k = 2$. They also introduced an "order-respecting" variant of k-FTM, called k-OFTM, where inputs are restricted in some natural way. They proposed an online algorithm and showed that its competitive ratio is at most $\frac{2kB}{\lfloor B/k \rfloor} + k$ for any $B \geq k$, where B is the size of the buffer. They also gave a lower bound of $\frac{B}{\lfloor 2B/k \rfloor}$ for deterministic online algorithms when $2B \geq k$ and k is a power of 2.

In this paper, we improve upper and lower bounds on the competitive ratio of k-OFTM. Our main result is to improve an upper bound of $O(k^2)$ by Kesselman et al. to $\frac{5B+\lfloor B/k \rfloor -4}{\lfloor B/2k \rfloor} = O(k)$ for $B \geq 2k$. Note that this upper bound is tight up to a multiplicative constant factor since the lower bound given by Kesselman et al. is $\Omega(k)$. We also give two lower bounds. First we give a lower bound of $\frac{2B}{\lfloor B/(k-1) \rfloor} + 1$ on the competitive ratio of deterministic online algorithms for any $k \geq 2$ and any $B \geq k-1$, which improves the previous lower bound of $\frac{B}{\lfloor 2B/k \rfloor}$ by a factor of almost four. Next, we present the first nontrivial lower bound on the competitive ratio of randomized algorithms. Specifically, we give a lower bound of $k-1$ against an oblivious adversary for any $k \geq 3$ and any B. Since a deterministic algorithm, as mentioned above, achieves an upper bound of about $10k$, this indicates that randomization does not help too much.

⋆ The author is supported in part by JSPS KAKENHI Grant Number 23700001.
⋆⋆ The author is supported by Funding Program for World-Leading Innovative R&D on Science and Technology (First Program).
⋆⋆⋆ The author is supported in part by JSPS KAKENHI Grant Number 24500013.

1 Introduction

When transmitting data through the Internet, each data is fragmented into smaller pieces, and such pieces are encapsulated into data packets. Packets are transmitted to the receiver via several switches and routers over a network, and are reconstructed into the original data at the receiver's side. One of the bottlenecks in achieving high throughput is processing ability of switches and routers. If the arrival rate of packets exceeds the processing rate of a switch, some packets must be dropped. To ease this inconvenience, switches are usually equipped with FIFO buffers that temporarily store packets which will be processed later. In this case, the efficiency of buffer management policies is important since it affects the performance of the overall network.

Aiello et al. [1] initiated the analysis of buffer management problem using the *competitive analysis* [10,32]: An input of the problem is a sequence of events where each event is an arrival event or a send event. At an arrival event, one packet arrives at an input port of the buffer (FIFO queue). Each packet is of unit size and has a positive value that represents its priority. A buffer can store at most B packets simultaneously. At an arrival event, if the buffer is full, the new packet is rejected. If there is room for the new packet, an online algorithm determines whether to accept it or not without knowing the future events. At each send event, the packet at the head of the queue is transmitted. The gain of an algorithm is the sum of the values of the transmitted packets, and the goal of the problem is to maximize it. If, for any input σ, the gain of an online algorithm ALG is at least $1/c$ of the gain of an optimal offline algorithm for σ, then we say that ALG is c-competitive.

Following the work of Aiello et al. [1], there has been a great amount of work related to the competitive analysis of buffer management. For example, Andelman et al. [5] generalized the two-value model of [1] into the multi-value model in which the priority of packets can take arbitrary values. Another generalization is to allow *preemption*, i.e., an online algorithm can discard packets existing in the buffer. Results of the competitiveness on these models are given in [18,33,20,4,3,12]. Also, management policies not only for a single queue but also for the whole switch are extensively studied, which includes multi-queue switches [7,5,2,6,28,9], shared-memory switches [14,19,27], CIOQ switches [21,8,25,22], and crossbar switches [23,24]. See [13] for a comprehensive survey.

Kesselman et al. [26] proposed another natural extension, called the *k-frame throughput maximization* problem (k-FTM), motivated by a scenario of reconstructing the original data from data packets at the receiver's side. In this model, a unit of data, called a *frame*, is fragmented into k packets (where the jth packet of the frame is called a j-packet for $j \in [1, k]$) and transmitted through the Internet. At the receiver's side, if all the k packets (i.e., the j-packet of the frame for all j) are received, the frame can be reconstructed (in such a case, we say that the frame is *completed*); otherwise, even if one of them is missing, the receiver can obtain nothing. The goal is to maximize the number of completed frames. Kesselman et al. [26] considered this scenario on a single FIFO queue. They first showed that the competitive ratio of any deterministic algorithm for

k-FTM is unbounded even when $k = 2$ (which can also be applied to randomized algorithms with a slight modification). However, their lower bound construction somehow deviates from the real-world situation, that is, although each packet generally arrives in order of departure in a network such as a TCP/IP network, in their adversarial input sequence the 1-packet of the frame f_i arrives prior to that of the frame $f_{i'}$, while the 2-packet of $f_{i'}$ arrives before that of f_i. Motivated by this, they introduced a natural setting for the input sequence, called the *order-respecting* adversary, in which, roughly speaking, the arrival order of the j-packets of f_i and $f_{i'}$ must obey the arrival order of the j'-packets of f_i and $f_{i'}$ ($j' < j$) (a formal definition will be given in Sec. 2). We call this restricted problem the *order-respecting k-frame throughput maximization* problem (k-OFTM). For k-OFTM, they showed that the competitive ratio of any deterministic algorithm is at least $B/\lfloor 2B/k \rfloor$ when $2B \geq k$ and k is a power of 2. As for an upper bound, they designed a non-preemptive algorithm called StaticPartitioning (SP), and showed that its competitive ratio is at most $\frac{2kB}{\lfloor B/k \rfloor} + k$ for any $B \geq k$.

1.1 Our Results

In this paper, we present the following results:

(i) We design a deterministic algorithm Middle-Drop and Flush (MF) for $B \geq 2k$, and show that its competitive ratio is at most $\frac{5B+\lfloor B/k \rfloor - 4}{\lfloor B/2k \rfloor}$. Note that this ratio is $O(k)$, which improves $O(k^2)$ of Kesselman et al. [26] and matches the lower bound of $\Omega(k)$ up to a constant factor.

(ii) For any deterministic algorithm, we give a lower bound of $\frac{2B}{\lfloor B/(k-1) \rfloor} + 1$ on the competitive ratio for any $k \geq 2$ and any $B \geq k - 1$. This improves the previous lower bound of $\frac{B}{\lfloor 2B/k \rfloor}$ by a factor of almost four. Moreover, we show that the competitive ratio of any deterministic online algorithm is unbounded if $B \leq k - 2$.

(iii) In the randomized setting, we establish the first nontrivial lower bound of $k - 1$ against an oblivious adversary for any $k \geq 3$ and any B. This bound matches our deterministic upper bound mentioned in (i) up to a constant factor, which implies that randomization does not help for this problem.

Because of the space restriction, all the proofs of the lemmas and theorems are omitted and are included in [17].

1.2 Used Techniques

Let us briefly explain an idea behind our algorithm MF. The algorithm SP by Kesselman et al. [26] works as follows: (1) It virtually divides its buffer evenly into k subbuffers, each with size $A = \lfloor \frac{B}{k} \rfloor$, and each subbuffer (called *j-subbuffer* for $j \in [1, k]$) is used for storing only j-packets. (2) If the j-subbuffer overflows, i.e., if a new j-packet arrives when A j-packets are already stored in the j-subbuffer, it rejects the newly arriving j-packet (the "tail-drop" policy). It can be shown that SP behaves poorly when a lot of j-packets arrive at a burst, which increases SP's competitive ratio as bad as $\Omega(k^2)$ (such a bad example for

SP is included in the full version of this paper [17]). In this paper, we modify the tail-drop policy and employ the "middle-drop" policy, which preempts the $(\lfloor A/2 \rfloor + 1)$st packet in the j-subbuffer and accepts the newly arriving j-packet, which is crucial in improving the competitive ratio to $O(k)$, as explained in the following.

MF partitions the whole set of given frames into *blocks* $BL_1, BL_2, \ldots$, each with about $3B$ frames, using the rule concerning the arrival order of 1-packets. (This rule is explained in Sec. 3.1 at the definition of MF, where the block BL_i corresponds to the set of frames with the *block number* i.) Each block is categorized into *good* or *bad*: At the beginning of the input, all the blocks are good. At some moment during the execution of MF, if there is no more possibility of completing at least $\lfloor A/2 \rfloor$ frames of a block BL_i (as a result of preemptions and/or rejections of packets in BL_i), then BL_i turns bad. In such a case, MF completely gives up BL_i and preempts all the packets belonging to BL_i in its buffer if any (which is called the "flush" operation). Note that at the end of input, MF completes at least $\lfloor A/2 \rfloor$ frames of a good block.

Consider the moment when the block BL_i turns bad from good, which can happen only when preempting a j-packet p (for some j) of BL_i from the j-subbuffer. Due to the property of the middle-drop policy, we can show that there exist two integers i_1 and i_2 ($i_1 < i < i_2$) such that (i) just after this flush operation, BL_{i_1} and BL_{i_2} are good and all the blocks $BL_{i_1+1}, BL_{i_1+2}, \ldots, BL_{i_2-1}$ are bad, and (ii) just before this flush operation, all the j-packets of BL_i (including p) each of which belongs to a frame that still has a chance of being completed are located between p_1 and p_2, where p_1 and p_2 are j-packets in the buffer belonging to BL_{i_1} and BL_{i_2}, respectively. The above (ii) implies that even though i_2 may be much larger than i_1 (and hence there may be many blocks between BL_{i_1} and BL_{i_2}), the arrival times of p_1 and p_2 are close (since p_1 is still in the buffer when p_2 arrived). This means that j-packets of BL_{i_1} through BL_{i_2} arrived at a burst within a very short span, and hence any algorithm (even an optimal offline algorithm OPT) cannot accept many of them. In this way, we can bound the number of packets accepted by OPT (and hence the number of frames completed by OPT) between two consecutive good blocks. More precisely, if BL_{i_1} and BL_{i_2} are consecutive good blocks at the end of the input, we can show that the number of frames in $BL_{i_1}, BL_{i_1+1}, \ldots, BL_{i_2-1}$ completed by OPT is at most $5B + A - 4 = O(B)$ using (i). Recall that MF completes at least $\lfloor A/2 \rfloor = \Omega(B/k)$ frames of BL_{i_1} since BL_{i_1} is good, which leads to the competitive ratio of $O(k)$.

1.3 Related Results

In addition to the above mentioned results, Kesselman et al. [26] proved that for any B, the competitive ratio of a preemptive greedy algorithm for k-OFTM is unbounded when $k \geq 3$. They also considered offline version of k-FTM and proved the approximation hardness. Recently, Kawahara and Kobayashi [16] proved that the optimal competitive ratio of 2-OFTM is 3, which is achieved by a greedy algorithm.

Scalosub et al. [31] proposed a generalization of k-FTM, called the *max frame goodput* problem. In this problem, a set of frames constitute a *stream*, and a constraint is imposed on the arrival order of packets within the same stream. They established an $O((kMB + M)^{k+1})$-competitive deterministic algorithm, where M denotes the number of streams. Furthermore, they showed that the competitive ratio of any deterministic algorithm is $\Omega(kM/B)$.

Emek et al. [11] introduced the *online set packing* problem. This problem is different from k-FTM in that each frame may consist of different number (at most $k_{\max}$) of packets. Also, a frame f consisting of $s(f)$ packets can be reconstructed if $s(f)(1 - \beta)$ packets are transmitted, where β $(0 \leq \beta < 1)$ is a given parameter. There is another parameter c representing the capacity of a switch. At an arrival event, several packets arrive at an input port of the queue. A switch can transmit c of them instantly, and operates a buffer management algorithm for the rest of the packets, that is, decides whether to accept them (if any). Emek et al. designed a randomized algorithm Priority, and showed that it is $k_{\max}\sqrt{\sigma_{\max}}$-competitive when $\beta = 0$ and $B = 0$, where $\sigma_{\max}$ is the maximum number of packets arriving simultaneously. They also derived a lower bound of $k_{\max}\sqrt{\sigma_{\max}}(\log\log k/\log k)^2$ for any randomized algorithm. If the number of packets in any frame is exactly k, Mansour et al. [29] showed that for any β the competitive ratio of Priority is $8k\sqrt{\sigma_{\max}(1-\beta)/c}$. Moreover, some variants of this problem have been studied [15,30].

2 Model Description and Notation

In this section, we give a formal description of the *order-respecting k-frame throughput maximization* problem (k-OFTM). A *frame* f consists of k packets $p_1, \ldots, p_k$. We say that two packets p and q belonging to the same frame are *corresponding*, or *p corresponds to q*. There is one buffer (FIFO queue), which can store at most B packets simultaneously. An input is a sequence of *phases* starting from the 0th phase. The ith phase consists of the ith *arrival subphase* followed by the ith *delivery subphase*. At an arrival subphase, some packets arrive at the buffer, and the task of an algorithm is to decide for each arriving packet p, whether to *accept* p or *reject* p. An algorithm can also discard a packet p' existing in the current buffer in order to make space (in which case we say that the algorithm *preempts* p'). If a packet p is rejected or preempted, we say that p is *dropped*. If a packet is accepted, it is stored at the tail of the queue. Packets accepted at the same arrival subphase can be inserted into the queue in an arbitrary order. At a delivery subphase, the first packet of the queue is transmitted if the buffer is nonempty. For a technical reason, we consider only the inputs in which at least one packet arrives.

If a packet p arrives at the ith arrival subphase, we write $\mathrm{arr}(p) = i$. For any frame $f = \{p_1, \ldots, p_k\}$ such that $\mathrm{arr}(p_1) \leq \cdots \leq \mathrm{arr}(p_k)$, we call p_i the *i-packet* of f. Consider two frames $f_i = \{p_{i,1}, \ldots, p_{i,k}\}$ and $f_{i'} = \{p_{i',1}, \ldots, p_{i',k}\}$ such that $\mathrm{arr}(p_{i,1}) \leq \cdots \leq \mathrm{arr}(p_{i,k})$ and $\mathrm{arr}(p_{i',1}) \leq \cdots \leq \mathrm{arr}(p_{i',k})$. If for any j and j', $\mathrm{arr}(p_{i,j}) \leq \mathrm{arr}(p_{i',j})$ if and only if $\mathrm{arr}(p_{i,j'}) \leq \mathrm{arr}(p_{i',j'})$, then we say that f_i

and $f_{i'}$ are *order-respecting*. If any two frames in an input sequence σ are order-respecting, we say that σ is *order-respecting*. If all the packets constituting a frame f are transmitted, we say that f is *completed*, otherwise, f is *incompleted*. The goal of k-FTM is to maximize the number of completed frames. k-OFTM is k-FTM where inputs are restricted to order-respecting sequences.

For an input σ, the *gain* of an algorithm ALG is the number of frames completed by ALG and is denoted by $V_{ALG}(\sigma)$. If ALG is a randomized algorithm, the gain of ALG is defined as an expectation $\mathbb{E}[V_{ALG}(\sigma)]$, where the expectation is taken over the randomness inside ALG. If $V_{ALG}(\sigma) \geq V_{OPT}(\sigma)/c$ ($\mathbb{E}[V_{ALG}(\sigma)] \geq V_{OPT}(\sigma)/c$) for an arbitrary input σ, we say that ALG is *c-competitive*, where OPT is an optimal offline algorithm for σ. Without loss of generality, we can assume that OPT never preempts packets and never accepts a packet of an incompleted frame.

3 Upper Bound

In this section, we present our algorithm Middle-Drop and Flush (MF) and analyze its competitive ratio.

3.1 Algorithm

We first give notation needed to describe MF. Suppose that n packets $p_1, p_2, \dots, p_n$ arrive at MF's buffer at the ith arrival subphase. For each packet, MF decides whether to accept it or not one by one (in some order defined later). Let t_{p_j} denote the time when MF deals with the packet p_j, and let us call t_{p_j} the *decision time* of p_j. Hence if $p_1, p_2, \dots, p_n$ are processed in this order, we have that $t_{p_1} < t_{p_2} < \cdots < t_{p_n}$. (We assume that OPT also deals with p_j at the same time t_{p_j}, which makes the competitive analysis simpler.) Also, let us call the time when MF transmits a packet from the head of its buffer at the ith delivery subphase the *delivery time* of the ith delivery subphase. A decision time or a delivery time is called an *event time*, and any other moment is called a *non-event time*. Note that during the non-event time, the configuration of the buffer is unchanged. For any event time t, $t+$ denotes any non-event time between t and the next event time. Similarly, $t-$ denotes any non-event time between t and the previous event time.

Let ALG be either MF or OPT. For a non-event time t and a packet p of a frame f, we say that p is *valid* for ALG at t if ALG has not dropped any packet of f before t, i.e., f still has a chance of being completed. In this case we also say that the frame f is *valid* for ALG at t. Note that a completed frame is valid at the end of the input. For a j-packet p and a non-event time t, if p is stored in MF's buffer at t, we define $\ell(t, p)$ as "1+(the number of j-packets located in front of p)", that is, p is the $\ell(t, p)$th j-packet in MF's queue. If p has not yet arrived at t, we define $\ell(t, p) = \infty$.

During the execution, MF virtually runs the following greedy algorithm GR_1 on the same input sequence. Roughly speaking, GR_1 is greedy for only 1-packets and ignores all $j(\geq 2)$-packets. Formally, GR_1 uses a FIFO queue of the same size B. At an arrival of a packet p, GR_1 rejects it if it is a j-packet for $j \geq 2$. If p is a 1-packet, GR_1 accepts it whenever there is a space in the queue. At a delivery subphase, GR_1 transmits the first packet of the queue as usual.

MF uses two internal variables `Counter` and `Block`. `Counter` is used to count the number of packets accepted by GR_1 modulo $3B$. `Block` takes a positive integer value; it is initially one and is increased by one each time `Counter` is reset to zero.

Define $A = \lfloor B/k \rfloor$. MF stores at most A j-packets for any j. For $j = 1$, MF refers to the behavior of GR_1 in the following way: Using two variables `Counter` and `Block`, MF divides 1-packets accepted by GR_1 into blocks according to their arrival order, each with $3B$ 1-packets. MF accepts the first A packets of each block and rejects the rest. For $j \geq 2$, MF ignores j-packets that are not valid. When processing a valid j-packet p, if MF already has A j-packets in its queue, then MF preempts the one in the "middle" among those j-packets and accepts p.

For a non-event time t, let $b(t)$ denote the value of `Block` at t. For a packet p, we define the *block number* $g(p)$ of p as follows. For a 1-packet p, $g(p) = b(t-)$ where t is the decision time of p, and for some $j(\geq 2)$ and a j-packet p , $g(p) = g(p')$ where p' is the 1-packet corresponding to p. Hence, all the packets of the same frame have the same block number. We also define the block number of frames in a natural way, namely, the block number $g(f)$ of a frame f is the (unique) block number of the packets constituting f. For a non-event time t and a positive integer u, let $h_{ALG,u}(t)$ denote the number of frames f valid for ALG at t such that $g(f) = u$.

Recall that at an arrival subphase, more than one packet may arrive at a queue. MF processes the packets ordered non-increasingly first by their frame indices and then by block numbers. If both are equal, they are processed in arbitrary order. That is, MF processes these packets by the following rule: Consider an i-packet p and an i'-packet p'. If $i < i'$, p is processed before p' and if $i' < i$, p' is processed before p. If $i = i'$, then p is processed before p' if $g(p) < g(p')$ and p' is processed before p if $g(p') < g(p)$. If $i = i'$ and $g(p) = g(p')$, the processing order is arbitrary. The formal description of MF is as follows.

Middle-Drop and Flush

```
Initialize: Counter := 0, Block := 1.
Let $p$ be a $j$-packet to be processed.
Case 1: $j = 1$:
  Case 1.1: If $GR_1$ rejects $p$, reject $p$.
  Case 1.2: If $GR_1$ accepts $p$, set Counter := Counter +1 and do the following.
    Case 1.2.1: If Counter $\leq A$, accept $p$. (We can guarantee that $MF$'s buffer
        has a space whenever Counter $\leq A$, as proven in [17].)
    Case 1.2.2: If $A <$ Counter $< 3B$, reject $p$.
    Case 1.2.3: If Counter $= 3B$, reject $p$ and set Counter := 0 and
        Block := Block + 1.
```

```
Case 2: $j \geq 2$:
  Case 2.1: If $p$ is not valid for $MF$ at $t_p-$, reject $p$.
  Case 2.2: If $p$ is valid for $MF$ at $t_p-$, do the following.
    Case 2.2.1: If the number of $j$-packets in $MF$'s buffer at $t_p-$ is at most
        $A-1$, accept $p$.
    Case 2.2.2: If the number of $j$-packets in $MF$'s buffer at $t_p-$ is at least $A$,
        then preempt the $j$-packet $p'$ such that $\ell(t_p-, p') = \lfloor A/2 \rfloor + 1$,
        and accept $p$. Preempt all the packets corresponding to $p'$ (if any).
      Case 2.2.2.1: If $h_{MF,g(p')}(t_p-) \leq \lfloor A/2 \rfloor$, preempt all the packets $p''$
          in $MF$'s buffer such that $g(p'') = g(p')$. (Call this operation "flush".)
      Case 2.2.2.2: If $h_{MF,g(p')}(t_p-) \geq \lfloor A/2 \rfloor + 1$, do nothing.
```

3.2 Overview of the Analysis

Let τ be any fixed time after MF processes the final event, and let c denote the value of `Counter` at τ. Also, we define $M = b(\tau) - 1$ if $c = 0$ and $M = b(\tau)$ otherwise. Note that for any frame f, $1 \leq g(f) \leq M$. Define the set G of integers as $G = \{M\} \cup \{i \mid$ there are at least $\lfloor A/2 \rfloor$ frames f completed by MF such that $g(f) = i\}$ and let $m = |G|$. For each $j \in [1, m]$, let a_j be the jth smallest integer in G. We call a block number *good* if it is in G and *bad* otherwise. Note that a_j denotes the jth good block number, and in particular that $a_m = M$ since $M \in G$. Our first key lemma is the following:

Lemma 1. $a_1 = 1$.

Since at the end of the input any valid frame is completed, we have $V_{OPT}(\sigma) = \sum_{i=1}^{M} h_{OPT,i}(\tau)$ and $V_{MF}(\sigma) = \sum_{i=1}^{M} h_{MF,i}(\tau) \geq \sum_{i=1}^{m} h_{MF,a_i}(\tau)$.

We first bound the gain of MF for good block numbers, which follows from the definition of G:

$$h_{MF,a_i}(\tau) \geq \lfloor A/2 \rfloor \text{ for any } i \in [1, m-1]. \quad (1)$$

We next focus on the mth good block number M. Since it has some exceptional properties, we discuss the number of completed frames with block number M independently of the other good block numbers as follows:

Lemma 2. *(a) If either $c = 0$ or $c \in [\lfloor A/2 \rfloor, 3B-1]$, $h_{MF,M}(\tau) \geq \lfloor A/2 \rfloor$. (b) If $c \in [1, \lfloor A/2 \rfloor - 1]$ and $M \geq 2$, $h_{MF,M}(\tau) + B - 1 \geq h_{OPT,M}(\tau)$. (c) If $c \in [1, \lfloor A/2 \rfloor - 1]$ and $M = 1$, $h_{MF,M}(\tau) \geq h_{OPT,M}(\tau)$.*

Also, we evaluate the number of OPT's completed frames from a viewpoint of good block numbers:

Lemma 3. *(a) $h_{OPT,M}(\tau) \leq 4B - 1$. (b) $\sum_{j=a_1}^{a_2-1} h_{OPT,j}(\tau) \leq 4B + A - 3$. (c) $\sum_{j=a_i}^{a_{i+1}-1} h_{OPT,j}(\tau) \leq 5B + A - 4$ for any $i \in [2, m-1]$.*

Using the above inequalities, we can obtain the competitive ratio of MF by case analysis on the values of M and c. First, note that if $M = 1$ then $c \geq 1$ because at least one packet arrives. Thus $V_{OPT}(\sigma) > 0$. Now if $M = 1$ and $c \in [1, \lfloor A/2 \rfloor - 1]$, then $\frac{V_{OPT}(\sigma)}{V_{MF}(\sigma)} = \frac{h_{OPT,1}(\tau)}{h_{MF,1}(\tau)} \leq 1$ by Lemma 2 (c). If $M = 1$ and $c \in [\lfloor A/2 \rfloor, 3B - 1]$, then $\frac{V_{OPT}(\sigma)}{V_{MF}(\sigma)} = \frac{h_{OPT,1}(\tau)}{h_{MF,1}(\tau)} \leq \frac{4B-1}{\lfloor A/2 \rfloor} < \frac{5B+A-4}{\lfloor A/2 \rfloor}$ by Lemma 2(a) and Lemma 3(a).

If $M \geq 2$ and $c \in \{0\} \cup [\lfloor A/2 \rfloor, 3B - 1]$,

$$\begin{aligned} V_{OPT}(\sigma) = \sum_{i=1}^{M} h_{OPT,i}(\tau) &= \sum_{i=1}^{m-1} \sum_{j=a_i}^{a_{i+1}-1} h_{OPT,j}(\tau) + h_{OPT,a_m}(\tau) \\ &\leq (m-1)(5B + A - 4) - B + 1 + (4B - 1) < m(5B + A - 4) \end{aligned}$$

by Lemma 3 (note that $a_1 = 1$ by Lemma 1 and $a_m = M$). Also, $V_{MF}(\sigma) \geq \sum_{i=1}^{m} h_{MF,a_i}(\tau) \geq m\lfloor A/2 \rfloor$ by (1) and Lemma 2(a). Therefore, $\frac{V_{OPT}(\sigma)}{V_{MF}(\sigma)} < \frac{5B+A-4}{\lfloor A/2 \rfloor}$. Finally, if $M \geq 2$ and $c \in [1, \lfloor A/2 \rfloor - 1]$,

$$\begin{aligned} V_{OPT}(\sigma) = \sum_{i=1}^{M} h_{OPT,i}(\tau) &= \sum_{i=1}^{m-1} \sum_{j=a_i}^{a_{i+1}-1} h_{OPT,j}(\tau) + h_{OPT,a_m}(\tau) \\ &\leq (m-1)(5B + A - 4) - B + 1 + h_{OPT,M}(\tau) \\ &\leq (m-1)(5B + A - 4) + h_{MF,M}(\tau) \end{aligned}$$

by Lemma 2(b) and Lemma 3(b) and (c). Also, $V_{MF}(\sigma) = \sum_{i=1}^{m} h_{MF,a_i}(\tau) \geq (m-1)\lfloor A/2 \rfloor + h_{MF,M}(\tau)$ by (1). Therefore,

$$\frac{V_{OPT}(\sigma)}{V_{MF}(\sigma)} \leq \frac{(m-1)(5B + A - 4) + h_{MF,M}(\tau)}{(m-1)\lfloor A/2 \rfloor + h_{MF,M}(\tau)} < \frac{5B + A - 4}{\lfloor A/2 \rfloor}.$$

We have proved that in all the cases $\frac{V_{OPT}(\sigma)}{V_{MF}(\sigma)} < \frac{5B+A-4}{\lfloor A/2 \rfloor}$. By noting that $\frac{5B+A-4}{\lfloor A/2 \rfloor} = \frac{5B+\lfloor B/k \rfloor-4}{\lfloor B/2k \rfloor}$, we have the following theorem:

Theorem 1. *When $B/k \geq 2$, the competitive ratio of MF is at most $\frac{5B+\lfloor B/k \rfloor-4}{\lfloor B/2k \rfloor}$.*

4 Lower Bound for Deterministic Algorithms

In this section, we give a lower bound on the competitive ratio for deterministic algorithms, improving the previous lower bound by a constant factor.

Theorem 2. *Suppose that $k \geq 2$. The competitive ratio of any deterministic algorithm is at least $\frac{2B}{\lfloor B/(k-1) \rfloor} + 1$ if $B \geq k - 1$, and unbounded if $B \leq k - 2$.*

5 Lower Bound for Randomized Algorithms

As for randomized algorithms, we give a first nontrivial lower bound. As mentioned previously, this matches the upper bound we proved in Sec. 3.2 up to a constant factor, implying that randomization does not help too much.

Theorem 3. *When $k \geq 3$, the competitive ratio of any randomized algorithm is at least $k - 1 - \epsilon$ for any constant ϵ against an oblivious adversary.*

References

1. Aiello, W., Mansour, Y., Rajagopolan, S., Rosén, A.: Competitive queue policies for differentiated services. Journal of Algorithms 55(2), 113–141 (2005)
2. Albers, S., Schmidt, M.: On the performance of greedy algorithms in packet buffering. SIAM Journal on Computing 35(2), 278–304 (2005)
3. Andelman, N.: Randomized queue management for DiffServ. In: In Proc. of the 17th ACM Symposium on Parallel Algorithms and Architectures, pp. 1–10 (2005)
4. Andelman, N., Mansour, Y.: Competitive management of non-preemptive queues with multiple values. In: Fich, F.E. (ed.) DISC 2003. LNCS, vol. 2848, pp. 166–180. Springer, Heidelberg (2003)
5. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies for QoS switches. In: Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 761–770 (2003)
6. Azar, Y., Litichevskey, A.: Maximizing throughput in multi-queue switches. Algorithmica 45(1), 69–90 (2006)
7. Azar, Y., Richter, Y.: Management of multi-queue switches in QoS networks. Algorithmica 43(1-2), 81–96 (2005)
8. Azar, Y., Richter, Y.: An improved algorithm for CIOQ switches. ACM Transactions on Algorithms 2(2), 282–295 (2006)
9. Bienkowski, M.: An optimal lower bound for buffer management in multi-queue switches. In: Proc. of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 1295–1305 (2010)
10. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
11. Emek, Y., Halldórsson, M., Mansour, Y., Patt-Shamir, B., Radhakrishnan, J., Rawitz, D.: Online set packing and competitive scheduling of multi-part tasks. In: Proc. of the 29th ACM Symposium on Principles of Distributed Computing, pp. 440–449 (2010)
12. Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in QoS switches. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 352–363. Springer, Heidelberg (2006)
13. Goldwasser, M.: A survey of buffer management policies for packet switches. ACM SIGACT News 41(1), 100–128 (2010)
14. Hahne, E., Kesselman, A., Mansour, Y.: Competitive buffer management for shared-memory switches. In: Proc. of the 13th ACM Symposium on Parallel Algorithms and Architectures, pp. 53–58 (2001)
15. Halldórsson, M., Patt-Shamir, B., Rawitz, D.: Online Scheduling with Interval Conflicts. In: Proc. of the 28th Symposium on Theoretical Aspects of Computer Science, pp. 472–483 (2011)
16. Kawahara, J., Kobayashi, K.M.: Optimal Buffer Management for 2-Frame Throughput Maximization. In: Proc. of the 20th International Colloquium on Structural Information and Communication Complexity (2013)
17. Kawahara, J., Kobayashi, K.M., Miyazaki, S.: Better Bounds for Online k-Frame Throughput Maximization in Network Switches. arXiv:1309.4919 [cs.DS] (2013)
18. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM Journal on Computing 33(3), 563–583 (2004)
19. Kesselman, A., Mansour, Y.: Harmonic buffer management policy for shared memory switches. Theoretical Computer Science 324(2-3), 161–182 (2004)

20. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 361–372. Springer, Heidelberg (2003)
21. Kesselman, A., Rosén, A.: Scheduling policies for CIOQ switches. Journal of Algorithms 60(1), 60–83 (2006)
22. Kesselman, A., Rosén, A.: Controlling CIOQ switches with priority queuing and in multistage interconnection networks. Journal of Interconnection Networks 9(1/2), 53–72 (2008)
23. Kesselman, A., Kogan, K., Segal, M.: Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing. In: Proc. of the 27th ACM Symposium on Principles of Distributed Computing, pp. 335–344 (2008)
24. Kesselman, A., Kogan, K., Segal, M.: Best effort and priority queuing policies for buffered crossbar switches. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 170–184. Springer, Heidelberg (2008)
25. Kesselman, A., Kogan, K., Segal, M.: Improved competitive performance bounds for CIOQ switches. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 577–588. Springer, Heidelberg (2008)
26. Kesselman, A., Patt-Shamir, B., Scalosub, G.: Competitive buffer management with packet dependencies. In: Proc. of the 23rd IEEE International Parallel and Distributed Processing Symposium, pp. 1–12 (2009)
27. Kobayashi, K., Miyazaki, S., Okabe, Y.: A tight bound on online buffer management for two-port shared-memory switches. In: Proc. of the 19th ACM Symposium on Parallel Algorithms and Architectures, pp. 358–364 (2007)
28. Kobayashi, K., Miyazaki, S., Okabe, Y.: Competitive buffer management for multi-queue switches in QoS networks using packet buffering algorithms. In: Proc. of the 21st ACM Symposium on Parallel Algorithms and Architectures, pp. 328–336 (2009)
29. Mansour, Y., Patt-Shamir, B., Rawitz, D.: Overflow management with multipart packets. In: Proc. of the 31st IEEE Conference on Computer Communications, pp. 2606–2614 (2011)
30. Mansour, Y., Patt-Shamir, B., Rawitz, D.: Competitive router scheduling with structured data. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 219–232. Springer, Heidelberg (2012)
31. Scalosub, G., Marbach, P., Liebeherr, J.: Buffer management for aggregated streaming data with packet dependencies. In: Proc. of the 29th IEEE Conference on Computer Communications, pp. 1–5 (2010)
32. Sleator, D., Tarjan, R.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
33. Sviridenko, M.: A lower bound for online algorithms in the FIFO model (2001) (unpublished manuscript)

The Solvable Cases of a Scheduling Algorithm

Sam Walker[1] and Yakov Zinder[1]

University of Technology, Sydney

Abstract. When considering the NP-hard problem of scheduling precedence constrained tasks with preemptions on identical parallel machines with the goal of minimising the maximum lateness, approximation algorithms are commonly studied. It is desirable to characterise in some way the circumstances under which a given algorithm will provide an optimal solution. This paper considers a well-known scheduling algorithm called the Brucker-Garey-Johnson Algorithm, known to produce optimal schedules whenever the precedence constraints are in the form of in-trees. A new class of partial orders is presented and it is proved not only that the Brucker-Garey-Johnson Algorithm will solve every problem instance constrained by a partial order from that class but also that no larger class has this property.

Keywords: scheduling theory, solvable cases, precedence constraints, identical parallel machines, preemptions, maximum lateness.

1 Introduction

The paper is concerned with the problem of preemptive scheduling of partially ordered tasks on parallel identical machines with the criterion of maximum lateness. This problem can be described as follows. A finite set of tasks (jobs, operations) $N = \{1, 2, \ldots, n\}$ is to be processed on $m \geq 1$ identical parallel machines (processors). The processing of tasks begins at time $t = 0$. Each machine can process only one task at a time. The order in which tasks can be processed is restricted by precedence constraints – a transitive, antireflexive and antisymmetric relation on N. If task j precedes task g, denoted $j \rightarrow g$, then task g cannot be processed until task j has been completed. In this case g is called a successor of j and j is a predecessor of g.

In order to be completed, a task j requires p_j units of processing time. At any point in time, the processing of a task can be interrupted and resumed later on the same or a different machine. In order to be completed, each task $j \in N$ should receive in total p_j units of processing time. The processing of a task can be interrupted only a finite number of times.

For each feasible schedule σ and each $j \in N$, the completion time of task j in schedule σ will be denoted by $C_j(\sigma)$. The goal is to minimise the criterion of maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}\{C_j(\sigma) - d_j\},$$

where d_j is the due date associated with task j. In the three field notation, see for example [1] or [6], the considered problem is denoted by $P|prmp, prec|L_{max}$.

Several algorithms were developed for this problem, [4], [7], [10]. Since the problem is NP-hard [8], all these algorithms can solve only some particular cases of the considered problem. Thus, the Brucker-Garey-Johnson Algorithm originally presented in [2] for unit execution time tasks was modified in [4] for arbitrary tasks with preemptions. It was shown in [4] that this modified algorithm produces an optimal schedule for $P|prmp, in-tree|L_{max}$, where $in-tree$ indicates that the partially ordered set of tasks is an in-tree. A similar result holds for the modification of the Brucker-Garey-Johnson algorithm presented in [7]. Other algorithms include one presented in [10] as well as a modification of the algorithm originally presented in [3] for unit execution time tasks. None of the algorithms developed for the considered problem have an exhaustive description of all solvable cases, here known as the domain of an algorithm. The domain $\mathcal{D}$ of an algorithm can be thought of as the set of all partial orders S such that

1. if $S \in \mathcal{D}$ all problem instances with precedence constraints S will be solved by the algorithm and
2. if $S \notin \mathcal{D}$ there exists a problem instance with precedence constraints S that is not solved by the algorithm.

This paper completely describes the domain of the Brucker-Garey-Johnson Algorithm. This leads to a new class of partially ordered sets which include as a particular case for example in-trees.

2 Scheduling Using Priorities

Several algorithms for the maximum lateness problem modify the original due dates (see for example [2], [3], [4], [7], [9] and [10]). Therefore, it is convenient to consider the following objective function

$$G(\sigma) = \max_{j \in N}[C_j(\sigma) + \mu_j] \tag{1}$$

without specifying how values μ_j are calculated. The algorithm below, Algorithm P, constructs a schedule, say σ, by scheduling tasks j according to their priorities $p_j(t, \sigma) + \mu_j$, where $p_j(t, \sigma)$ is the remaining processing time of task j at time t in schedule σ. It has been described several times before, including in [10].

Algorithm P schedules tasks in several stages. Each stage corresponds to a point in time that will be referred to as a point of allocation. According to Step 1 the first point of allocation is $t_0 = 0$. Let $0 = t_0 < t_1 < \ldots < t_k$ be the points of allocation. For each t_i, $0 \leq i < k$, let A_i be the set of all tasks j such that $p_j(t_i, \sigma) > 0$ and $p_g(t_i, \sigma) = 0$ for all $g \to j$. In other words, A_i is the set of tasks that are available for processing at point t_i. Steps 2, 3, 4 and 5 split A_i into three sets: N_1, N_2 and N_3. The priority of each task in N_1 is strictly greater than the priority of any task in N_2 and N_3; the priority of all tasks in N_2 are equal, and this common priority is strictly greater than the priority of any task in N_3. Any of these sets can be empty.

Let $\Delta = t_{i+1} - t_i$. According to Step 7, between t_i and t_{i+1}, each $j \in N_1$ receives $v_j = \Delta$ units of processing time. Therefore the tasks constituting N_1 occupy $m_1 = |N_1|$ machines. Between t_i and t_{i+1}, each $g \in N_2$ receives $v_g = \frac{m - m_1}{|N_2|} \Delta$ units of processing time (see Step 7). Tasks constituting N_3 are not processed between t_i and t_{i+1}.

Step 6 calculates Δ using the following conditions. Conditions (2) and (3) guarantee that the remaining processing time of each task is not less than corresponding v_j. Condition (4) ensures that at t_{i+1} the priority of any task in N_1 is not less than the priority of tasks in N_2. Condition (5) states that at t_{i+1} the priority of any task in N_1 is not less than the priority of any task in N_3. Condition (6) specifies that at t_{i+1} the priority of any task in N_2 is not less than the priority of any task in N_3.

Step 8 uses as a subroutine the McNaughton's Algorithm described in [5]. This algorithm schedules k tasks with no precedence constraints and processing times v_j on parallel machines in time

$$\max\left\{\frac{1}{m}\sum_{j=1}^{n} v_j, \max_j \{v_j\}\right\},$$

and is thus used to schedule each interval $[t_i, t_{i+1}]$ separately.

Algorithm P

1. Set $i = 0$ and $t_i = 0$..
2. Partition A_i into sets $S_1, S_2, \ldots, S_b$ such that for any $1 \le e \le b$ and any $j \in S_e$ and $g \in S_e$,

   $$p_j(t_i, \sigma) + \mu_j = p_g(t_i, \sigma) + \mu_g,$$

   and for any $1 \le e < b$ and any $j \in S_e$ and $g \in S_{e+1}$,

   $$p_j(t_i, \sigma) + \mu_j > p_g(t_i, \sigma) + \mu_g.$$

3. If $|S_1| \le m$, then among all $a \le b$ such that

   $$\sum_{e=1}^{a} |S_e| \le m$$

   select the largest one, say h. Set $N_1 = \cup_{e=1}^{h} S_e$ and $m_1 = |N_1|$. Otherwise, set $m_1 = 0$, $h = 0$ and $N_1 = \emptyset$.
4. If $h = b$ or $m_1 = m$, then set $N_2 = \emptyset$. Otherwise, set $N_2 = S_{h+1}$.
5. Set $N_3 = A_i \setminus N_1 \setminus N_2$.
6. Find the largest Δ satisfying, for all $j \in N_1$, $g \in N_2$ and $q \in N_3$, the following inequalities:

   $$p_j(t_i, \sigma) \ge \Delta \tag{2}$$

$$p_g(t_i, \sigma) \geq \frac{m - m_1}{|N_2|} \varDelta \tag{3}$$

$$p_j(t_i, \sigma) \; - \varDelta \; + \; \mu_j \geq p_g(t_i, \sigma) \; - \; \frac{m - m_1}{|N_2|} \varDelta + \mu_g \tag{4}$$

$$p_j(t_i, \sigma) \; - \varDelta \; + \; \mu_j \geq p_q(t_i, \sigma) + \mu_q \tag{5}$$

$$p_g(t_i, \sigma) \; - \; \frac{m - m_1}{|N_2|} \varDelta + \mu_g \geq p_q(t_i, \sigma) + \mu_q \tag{6}$$

7. For all $j \in N_1$, set $v_j = \varDelta$. For all $j \in N_2$, set
$$v_j = \frac{m - m_1}{|N_2|} \varDelta.$$
8. Let $t_{i+1} = t_i + \varDelta$ and use McNaughton's Algorithm to schedule v_j units of time for each $j \in N_1 \cup N_2$ on the interval $[t_i, t_{i+1}]$.
9. Set $i = i + 1$. If $A_i \neq \emptyset$, go to Step 2. Otherwise, stop.

Note that, due to Step 6 of the preceding algorithm, $p_j(t_i, \sigma) + \mu_j \geq p_g(t_i, \sigma) + \mu_g$ implies $p_j(t_{i+1}, \sigma) + \mu_j \geq p_g(t_{i+1}, \sigma) + \mu_g$ for all $0 \leq i < k$ and all $j \in A_i$ and $g \in A_i$.

3 Schedule Structure

Different methods of computing μ's may result in different schedules produced by Algorithm P. Defining $K(j) = \{g : j \to g\}$, the Brucker-Garey-Johnson Algorithm calculates μ_j as specified below.

Brucker-Garey-Johnson Algorithm

1. For every task j such that $K(j) = \emptyset$, set $\mu_j = -d_j$.
2. Select some task j such that
   (a) μ_j has not been specified;
   (b) for each $g \in K(j)$, μ_g has been already specified.
3. Set
$$\mu_j = \max \left\{ -d_j, \max_{g \in K(j)} (p_g + \mu_g) \right\}.$$
   If all tasks j in N are assigned μ_j, then stop. Otherwise, go to Step 2.

This algorithm computes μ's such that

$$\mu_j \geq \mu_g + p_g, \qquad \text{for all pairs } j \to g. \tag{7}$$

The three lemmas below characterise μ's and the structure of the resultant schedule. These lemmas and their proofs can be found in [7] and [10].

Lemma 1. *For all schedules σ, $G(\sigma) = L_{max}(\sigma)$.*

Let σ^{BGJ} be a schedule produced by the Brucker-Garey-Johnson Algorithm. Consider all points in time t such that there exists a task $\tilde{\jmath}$ satisfying

$$C_{\tilde{\jmath}}\left(\sigma^{BGJ}\right) \geq t \qquad \text{and} \qquad t + p_{\tilde{\jmath}}\left(t, \sigma^{BGJ}\right) + \mu_{\tilde{\jmath}} = L_{max}\left(\sigma^{BGJ}\right). \tag{8}$$

Among all considered t, select the smallest one, say τ. It is easy to see that if $\tau = 0$, then σ^{BGJ} is an optimal schedule. In what follows, we assume that $t_{i^*-1} < \tau \leq t_{i^*}$, for some $i^* \geq 1$.

Lemma 2. *The set N_1 specified at time point t_{i^*-1} is empty.*

Lemma 3. *The set N_2 associated with t_{i^*-1} contains a task x such that*

$$C_x\left(\sigma^{BGJ}\right) \geq t_{i^*} \qquad \textit{and} \qquad t_{i^*} + p_x\left(t_{i^*}, \sigma^{BGJ}\right) + \mu_x = G\left(\sigma^{BGJ}\right).$$

4 A New Class of Partial Orders

It will now be proved that the domain of the Brucker-Garey-Johnson Algorithm is the set of all partial orders not containing a member from either one of two classes of prohibited graph, described below. A pair of nodes is said to be independent if neither node precedes the other. A set of more than two nodes is said to be independent if all nodes are pairwise independent.

The first class is that of graphs with six nodes, divided into two sets A_1 and B_1, each containing three independent nodes. One node in A_1, which will be called the key node, precedes all three nodes in B_1. Any precedence constraints between the other two nodes in A_1 and the three nodes in B_1 are allowed in this class of graphs.

The second class is that of graphs with eight nodes, divided into two sets A_2 and B_2, each containing four independent nodes. Two nodes in A_2, which will be called the key nodes, precede disjoint sets, each of two nodes, in B_2 in such a way that each precedes only one of these two sets. Any precedence constraints between the other two nodes in A_2 and the four nodes in B_2 are allowed in this class of graphs.

These classes of prohibited graphs are represented below, with the mandatory precedence constraints as solid lines and the optional constraints as dashed lines. The key nodes are shaded.

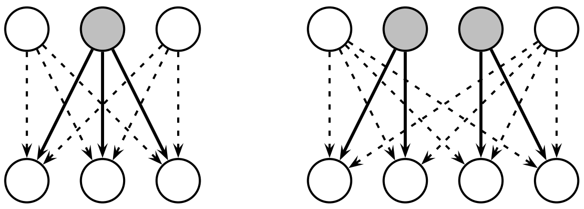


Fig. 1. The two classes of prohibited graphs

There are thirteen isomorphically distinct prohibited graphs in the first class. Several prohibited graphs in the second class contain a graph from the first class. Consequently thirty prohibited graphs are needed to specify the domain of the Brucker-Garey-Johnson Algorithm.

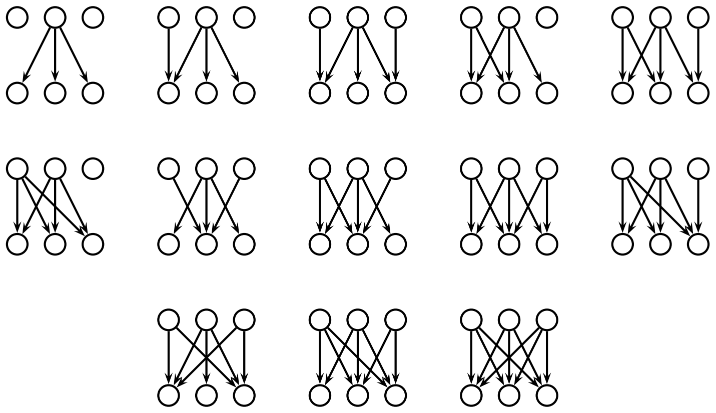


Fig. 2. The thirteen prohibited graphs of the first kind

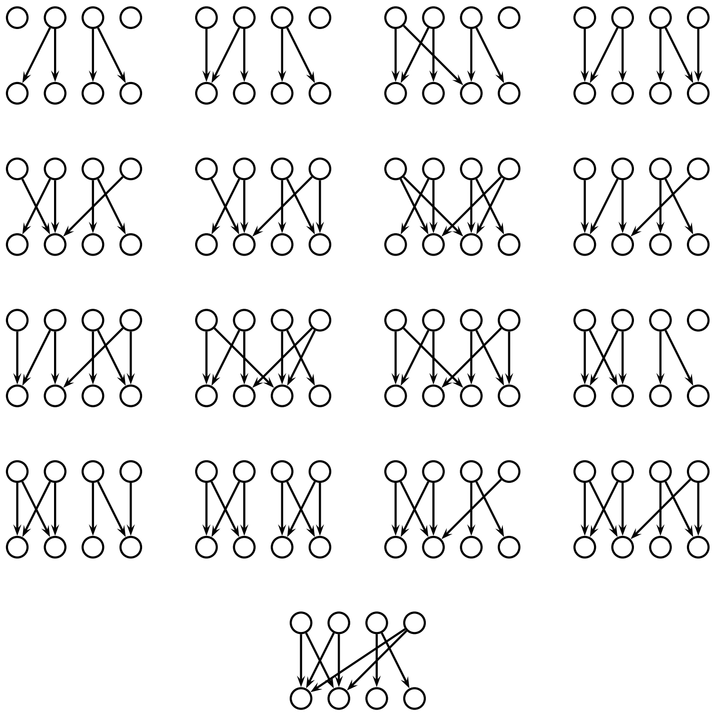


Fig. 3. The seventeen prohibited graphs of the second kind

The first thing to note is that all prohibited subgraphs contain a subgraph consisting of three nodes n_1, n_2 and n_3 with two order relations $n_1 \to n_2$ and $n_1 \to n_3$. Consequently all in-trees are in the domain of the Brucker-Garey-Johnson Algorithm, as was proven in [4]. Note, however, that many graphs that are not in-trees are also in the domain of the considered algorithm, indicating a broadening of the well known classical result. Additionally, to the authors' knowledge this is the first time the domain of an algorithm has been completely specified.

5 Proving Sufficiency

For the purpose of the following analysis it is convenient to introduce the following notation. For each $0 \le i < i^*$ define

$$H_i = \left\{ j : j \in A_i, p_j \left(t_{i+1}, \sigma^{BGJ} \right) + \mu_j \ge p_x \left(t_{i^*}, \sigma^{BGJ} \right) + \mu_x \right\} .$$

Lemma 4. *For all $0 \le i' < i < i^*$, if $j \in H_i$, then there exists a task $j' \in H_{i'}$ such that either $j = j'$ or $j' \to j$.*

Proof. If $j \in A_{i'}$ the relation

$$p_j \left(t_{i'+1}, \sigma^{BGJ} \right) + \mu_j \ge p_j \left(t_{i+1}, \sigma^{BGJ} \right) + \mu_j \ge p_x \left(t_{i^*}, \sigma^{BGJ} \right) + \mu_x$$

implies $j \in H_{i'}$.

On the other hand, if $j \notin A_{i'}$ the fact that $j \in H_i \subseteq A_i$ implies $C_j \left(\sigma^{BGJ} \right) > t_{i'}$ and thus there exists at least one task $j' \in A_{i'}$ such that $j' \to j$. In this case (7) implies

$$p_{j'} \left(t_{i'+1}, \sigma^{BGJ} \right) + \mu_{j'} \ge p_j \left(t_{i+1}, \sigma^{BGJ} \right) + \mu_j \ge p_x \left(t_{i^*}, \sigma^{BGJ} \right) + \mu_x$$

thus completing the proof. □

Corollary 1. *For all $0 \le i < i^*$, $|H_i| \ge 1$.*

Suppose there is some problem instance for which the Brucker-Garey-Johnson Algorithm does not provide the optimal solution. The lemma below provides an insight into the structure of the resulting schedule.

Lemma 5. *If there exists $0 \le \bar{\imath} < i^*$ such that $|H_i| \le m$ for $i < \bar{\imath}$ and $|H_i| \ge m$ for $i \ge \bar{\imath}$, σ^{BGJ} is an optimal schedule.*

Proof. For all $i < \bar{\imath}$, $|H_i| \le m$ and

$$p_j \left(t_i, \sigma^{BGJ} \right) + \mu_j > p_g \left(t_i, \sigma^{BGJ} \right) + \mu_g$$

for all $j \in H_i$ and all $g \in A_i \setminus H_i$. Consequently, $H_i \subseteq N_1$ in that iteration of Algorithm P. From Step 6 of Algorithm P the tasks in N_1 are processed during the entire interval $[t_i, t_{i+1}]$. Additionally, (7) implies that if $j \in H_i$, all $g \to j$ have

$$\mu_g \geq p_j + \mu_j \geq p_j \left(t_{i+1}, \sigma^{BGJ}\right) + \mu_j \geq p_x \left(t_{i^*}, \sigma^{BGJ}\right) + \mu_x$$

and thus $g \in A_{i'}$ implies $g \in H_{i'}$. As a consequence, for all $0 \leq i < \bar{\imath}$ and all $j \in H_i$ and $t \in [t_i, t_{i+1}]$ the relation $p_j \left(t, \sigma^{BGJ}\right) \leq p_j \left(t, \sigma\right)$ holds for all schedules σ.

For $\bar{\imath} \leq i < i^*$ all $j \in H_i$ were processed as much as possible before time $t_{\bar{\imath}}$. For all $t \in [t_{\bar{\imath}}, t_{i^*}]$ tasks with $p_j \left(t, \sigma^{BGJ}\right) + \mu_j \geq p_x \left(t_{i^*}, \sigma^{BGJ}\right) + \mu_x$ consume all available machine-time: consequently for every schedule σ there is a task j with $C_j(\sigma) \geq t_{i^*}$ such that $t_{i^*} + p_j(t_{i^*}, \sigma) + \mu_j \geq L_{max} \left(\sigma^{BGJ}\right)$. □

The corollary below follows from the lemma above and Corollary 1.

Corollary 2. *When $m = 1$ the Brucker-Garey-Johnson Algorithm produces an optimal schedule.*

Let the class of graphs not containing any of the thirty prohibited subgraphs described earlier be denoted $\mathcal{D}$.

Theorem 1. *For any problem with precedence constraints in $\mathcal{D}$, the Brucker-Garey-Johnson Algorithm produces an optimal schedule.*

Proof. By Lemma 2 and Lemma 5 the non-optimal schedule must have some $0 \leq \underline{i} < \hat{\imath} < \bar{\imath} < i^*$ such that $|H_{\hat{\imath}}| < m$, $|H_{\bar{\imath}}| > m$ and $|H_{\underline{i}}| > m$.

Of all tasks $j \in H_{\hat{\imath}}$ select as $\hat{\jmath}$ a task that precedes the largest number of tasks from $H_{\bar{\imath}}$. Since $|H_{\bar{\imath}}| \geq |H_{\hat{\imath}}| + 2$ task $\hat{\jmath}$ must have at least two successors in $H_{\bar{\imath}}$. From Lemma 4 either $j = \hat{\jmath}$ or $j \to \hat{\jmath}$ for some $j \in H_{\underline{i}}$, so at least one task $j \in H_{\underline{i}}$ must have at least two successors in $H_{\bar{\imath}}$. Select as $j_1 \in H_{\underline{i}}$ the task with the most such successors. From Corollary 2 a problem instance creating a non-optimal schedule must have $m \geq 2$, which implies $|H_{\bar{\imath}}| \geq 3$ and $|H_{\underline{i}}| \geq 3$. Consequently, if j_1 has three or more successors in $H_{\bar{\imath}}$ then the problem instance contains a prohibited subgraph of the first kind, with j_1 as the key task.

On the other hand, if j_1 precedes two tasks in $H_{\bar{\imath}}$ this implies $\hat{\jmath}$ also precedes two tasks in $H_{\bar{\imath}}$. Since $|H_{\bar{\imath}}| \geq |H_{\hat{\imath}}| + 2$ there must exist $\hat{\jmath}' \in H_{\hat{\imath}} \setminus \{\hat{\jmath}\}$ that precedes two tasks in $H_{\bar{\imath}} \setminus K\left(\hat{\jmath}\right)$. This implies that $|H_{\underline{i}}| > m > |H_{\hat{\imath}}| \geq |\{\hat{\jmath}, \hat{\jmath}'\}| = 2$ and thus $H_{\underline{i}} \geq 4$. Again from Lemma 4 either $j = \hat{\jmath}'$ or $j \to \hat{\jmath}'$ for some $j \in H_{\underline{i}} \setminus \{j_1\}$. Let $j_2 \in H_{\underline{i}} \setminus \{j_1\}$ be such a task. Since $H_{\underline{i}} \geq 4$, and since $|K(j_1) \cap H_{\bar{\imath}}| = 2$, $|K(j_2) \cap H_{\bar{\imath}}| = 2$ and $K(j_1) \cap K(j_2) \cap H_{\bar{\imath}} = \emptyset$, this problem instance contains a prohibited subgraph of the second kind, with j_1 and j_2 as the key tasks. □

6 Proving Necessity

I what follows $O(\varepsilon)$ will denote any function with the property

$$\lim_{\varepsilon \to 0^+} \frac{O(\varepsilon)}{\varepsilon} = \text{constant}.$$

Theorem 2. *For any partial order not in $\mathcal{D}$ there exists an assignment of m, p_i and d_i such that the Brucker-Garey-Johnson Algorithm produces a non-optimal schedule.*

Proof. The proof will be accomplished by taking any partial order of tasks with one of the prohibited subgraphs and assigning due dates and processing times such that the Brucker-Garey-Johnson Algorithm will schedule it incorrectly.

Suppose that the given partially ordered set of tasks contains a prohibited subgraph of the first kind, composed of the set of tasks M with key task k. In order to specify an instance for which the Brucker-Garey-Johnson Algorithm fails to construct an optimal schedule, set $m = 2$, $p_k = 2$, $p_j = 1$ for all $j \in M \setminus \{k\}$, and $p_j = \varepsilon$ for all $j \in N \setminus M$. To complete the instance, set $d_k = 2$, $d_j = 3\frac{1}{3}$ for all $j \in K(k) \cap M$, $d_j = 1$ for all $j \in M \setminus (\{k\} \cup K(k))$, and $d_j = \delta$ for all $j \in N \setminus M$.

The value of ε may be made arbitrarily small, and δ may be made arbitrarly large, reducing the impact of tasks in $N \setminus M$ on the resulting schedule. The Brucker-Garey-Johnson Algorithm will first schedule, in some number of intervals, all tasks preceding $M \setminus K(k)$. Since they are more urgent than any task in M they will all be completed by time $n\varepsilon$. At this point in time, the three tasks constituting $M \setminus K(k)$ may have been processed for different amounts, and thus have priorities differing by $O(\varepsilon)$. The more urgent tasks, i.e. the tasks receiving less processing time, will be processed faster than the less urgent tasks according to Algorithm P and the priorities will converge at a linear rate, therefore at some time $O(\varepsilon)$ all tasks in $M \setminus K(k)$ will have received the same amount of processing. These tasks will be processed until time $1\frac{1}{2} + O(\varepsilon)$ at which point the two tasks in $M \setminus (K(k) \cup \{k\})$ will simultaneously be completed. After this task k will be processed for one unit of processing time on the interval $[1\frac{1}{2} + O(\varepsilon), 2\frac{1}{2} + O(\varepsilon)]$ and be completed. During this interval, any tasks in $N \setminus (M \cup K(k))$ that have not already been completed will be processed in parallel with k. Within time $n\varepsilon$ all tasks in $K(k) \setminus M$ that precede any task in M will be processed, resulting in the three tasks in $M \cap K(k)$ having priorities that differ by $O(\varepsilon)$. As before, after some amount of time $O(\varepsilon)$, these three tasks will again have equal priority. These tasks will be completed in the interval $[2\frac{1}{2} + O(\varepsilon), 4 + O(\varepsilon)]$. After this, all remaining tasks will be processed in $O(\varepsilon)$ time.

For adequately large δ and adequately small ε the maximum lateness of this schedule is $\frac{2}{3} + O(\varepsilon)$, attained on tasks in set $K(k) \cap M$.

An alternative schedule would begin with an interval $[0, n\varepsilon]$ in which all tasks not in M and not preceded by a task in M are completed. On interval $\left[n\varepsilon, \frac{1}{3} + n\varepsilon\right]$ task k receives $\frac{1}{3}$ of a unit of processing time while the two tasks in $M \setminus (\{k\} \cup K(k))$ each receive $\frac{1}{6}$ of a unit of processing time. On interval $\left[\frac{1}{3} + n\varepsilon, 1\frac{7}{12} + n\varepsilon\right]$ all three tasks in $M \setminus K(k)$ receive $\frac{5}{6}$ of a unit of processing, finishing both tasks in $M \setminus (\{k\} \cup K(k))$. On interval $\left[1\frac{7}{12} + n\varepsilon, 2\frac{5}{12} + n\varepsilon\right]$ task k is processed for $\frac{5}{6}$ of a unit, and completed. On interval $\left[2\frac{5}{12} + n\varepsilon, 2\frac{5}{12} + 2n\varepsilon\right]$ all tasks in $N \setminus M$ that may be completed in this interval are completed. On interval $\left[2\frac{5}{12} + 2n\varepsilon, 3\frac{11}{12} + 2n\varepsilon\right]$ all three tasks in $M \cap K(k)$ receive 1 unit of processing and are completed. Finally, all remaining tasks in $N \setminus M$ are processed to completion in interval $\left[3\frac{11}{12} + 2n\varepsilon, 3\frac{11}{12} + 3n\varepsilon\right]$.

The maximum lateness of this schedule, for adequately large δ and adequately small ε is $\frac{7}{12}+2n\varepsilon$ obtained from the tasks in set $K(k)\cap M$. Since ε can be made arbitrarily small there is a value of ε for which this schedule has a maximum lateness less than $\frac{2}{3}$, thus the Brucker-Garey-Johnson Algorithm fails to produce an optimal schedule.

On the other hand, suppose the given partially ordered set of tasks contains a prohibited subgraph of the second kind, composed of the set of tasks M with key tasks k_1 and k_2. In order to specify an instance for which the Brucker-Garey-Johnson Algorithm fails to construct an optimal schedule, set $m = 3$, $p_{k_1} = p_{k_2} = 2$, $p_j = 1$ for all $j \in M \setminus \{k_1, k_2\}$, and $p_j = \varepsilon$ for all $j \in N \setminus M$. To complete the instance, set $d_{k_1} = d_{k_2} = 2$, $d_j = 3\frac{1}{4}$ for all $j \in (K(k_1)\cup K(k_2))\cap M$, $d_j = 1$ for all $j \in M \setminus (\{k_1, k_2\} \cup K(k_1) \cup K(k_2))$, and $d_j = \delta$ for all $j \in N \setminus M$.

The Brucker-Garey-Johnson Algorithm will first schedule, in some number of intervals, all tasks preceding $M \setminus (K(k_1) \cup K(k_2))$. Since they are more urgent than any task in M they will all be complete by time $n\varepsilon$. At this point the tasks in $M \setminus (K(k_1) \cup K(k_2))$ may have been processed for different amounts, and thus have priorities differing by $O(\varepsilon)$. As before, the priorities will converge at a linear rate, therefore at some time $O(\varepsilon)$ all tasks in $M \setminus (K(k_1) \cup K(k_2))$ will have received the same amount of processing. These tasks will be processed until time $1\frac{1}{3} + O(\varepsilon)$ at which point the two tasks in $M \setminus (K(k_1) \cup K(k_2) \cup \{k_1, k_2\})$ will simultaneously be completed. After this tasks k_1 and k_2 will be processed for one unit of processing time on the interval $[1\frac{1}{3} + O(\varepsilon), 2\frac{1}{3} + O(\varepsilon)]$ and both will be completed. In parallel with this, any tasks in $N \setminus (M \cup K(k_1) \cup K(k_2))$ that have not already been completed will be processed in parallel with k_1 and k_2. Within time $n\varepsilon$ all tasks in $(K(k_1) \cup K(k_2)) \setminus M$ that precede any task in M will be processed, resulting in the four tasks in $M \cap (K(k_1) \cup K(k_2))$ having priorities that differ by $O(\varepsilon)$. As before, these tasks will have equal priority again after some amount of time $O(\varepsilon)$. These tasks will receive equal processing on the interval $[2\frac{1}{3} + O(\varepsilon), 3\frac{2}{3} + O(\varepsilon)]$ and be completed. After this, all remaining tasks will be processed in $O(\varepsilon)$ time.

For adequately large δ and adequately small ε the maximum lateness of this schedule is $\frac{5}{12} + O(\varepsilon)$, obtained from the tasks in set $(K(k_1) \cup K(k_2)) \cap M$.

An alternative schedule would begin with an interval $[0, n\varepsilon]$ in which all tasks not in M and not preceded by a task in M are completed. On interval $\left[n\varepsilon, \frac{1}{6} + n\varepsilon\right]$ tasks k_1 and k_2 receive processing time $\frac{1}{6}$ while the two tasks in $M \setminus (\{k_1, k_2\} \cup K(k_1) \cup K(k_2))$ each receive $\frac{1}{12}$. On interval $\left[\frac{1}{6} + n\varepsilon, 1\frac{7}{18} + n\varepsilon\right]$ all four tasks in $M \setminus (K(k_1) \cup K(k_2))$ receive processing time $\frac{11}{12}$, completing two of them. On interval $\left[1\frac{7}{18} + n\varepsilon, 2\frac{11}{36} + n\varepsilon\right]$ tasks k_1 and k_2 are processed for $\frac{11}{12}$ of a unit, completing them. On interval $\left[2\frac{11}{36} + n\varepsilon, 2\frac{11}{36} + 2n\varepsilon\right]$ all tasks in $N \setminus M$ that may be completed in this interval are completed. On interval $\left[2\frac{11}{36} + 2n\varepsilon, 3\frac{23}{36} + 2n\varepsilon\right]$ all four tasks in $(K(k_1) \cup K(k_2)) \cap M$ receive 1 unit of processing and are completed. Finally, all remaining tasks in $N \setminus M$ are processed to completion in interval $\left[3\frac{23}{36} + 2n\varepsilon, 3\frac{23}{36} + 3n\varepsilon\right]$.

The maximum lateness of this schedule, for adequately large δ and adequately small ε is $\frac{7}{18} + 2n\varepsilon$, and as before a small enough value of ε gives this a smaller maximum lateness than σ^{BGJ}. $\square$

References

1. Brucker, P.: Scheduling Algorithms. Springer, Berlin (2007)
2. Brucker, P., Garey, M.R., Johnson, D.S.: Scheduling Equal-Length Tasks Under Treelike Precedence Constraints to Minimize Maximum Lateness. Mathematics of Operations Research 2, 275–284 (1977)
3. Garey, M.R., Johnson, D.S.: Two Processor Scheduling with Start Times and Deadlines. SIAM Journal on Computing 6, 416–426 (1977)
4. Lawlwer, E.L.: Preemptive Scheduling of Precedence-Constrained Jobs on Parallel Machines. Deterministic and Stochastic Scheduling, NATO Advanced Study Institutes Series 84, 101–123 (1982)
5. McNaughton, R.: Scheduling with Deadlines and Loss Functions. Management Science 6, 1–12 (1959)
6. Pinedo, M.: Scheduling Theory, Algorithms, and Systems. Springer, New York (2008)
7. Singh, G., Zinder, Y.: Worst Case Performance of Critical Path Type Algorithms. International Transactions in Operations Research 7, 383–399 (2000)
8. Ullman, J.D.: NP-complete scheduling problems. Journal of Computer and System Sciences 10, 384–393 (1973)
9. Zinder, Y., Roper, D.: An Iterative Algorithm for Scheduling Unit-Time Tasks with Precedence Constraints to Minimise the Maximum Lateness. Annals of Operations Research 81, 321–340 (1998)
10. Zinder, Y., Singh, G.: Preemptive Scheduling on Parallel Processors with Due Dates. Asia-Pacific Journal of Operations Research 22, 445–462 (2005)

Exact Sublinear Binomial Sampling[⋆]

Martín Farach-Colton and Meng-Tsung Tsai

Rutgers University, New Brunswick NJ 08901, USA
{farach,mtsung.tsai}@cs.rutgers.edu

Abstract. Drawing a random variate from a given binomial distribution $B(n,p)$ is an important subroutine in many large-scale simulations. The naive algorithm takes $\mathcal{O}(n)$ time and has no precision loss, however, this method is often too slow in many settings. The problem of sampling from a binomial distribution in sublinear time has been extensively studied and implemented in such packages as R [22] and the GNU Scientific Library (GSL) [10], however, all known sublinear-time algorithms involve precisions loss, which introduces artifacts into the sampling, such as discontinuities.

In this paper, we present the first algorithm, to the best of our knowledge, that samples binomial distributions in sublinear time with no precision loss.

1 Introduction

Let $B(n,p)$ be the binomial distribution of n trials and success rate p. Drawing a random variate b from $B(n,p)$ means that

$$\Pr[b=k] = p^k(1-p)^{n-k}\binom{n}{k} \text{ for all } k \in \{0,1,\ldots,n\}. \tag{1}$$

To draw a random variate b from a binomial distribution $B(n,p)$, one can naively realize n Bernoulli trials of success rate p and count how many of them have a positive outcome in $O(n)$ time. In other words, binomial sampling can be used as an alternative for realizing n Bernoulli trials.

Sampling variates from binomial distribution is a common procedure provided by the GNU Scientific Library [10] and the statistical software R [22], both of which use the algorithm BTPE proposed in [14]. These implementations have inaccuracies, such as the discontinuity shown in Figure 1. In particular, we show that BTPE substantially overestimates the probability of the tail of the distribution. In Figure 1, the overestimation is by a factor of 2.59, or 0.74% of the total samples. Thus, the occurence of a rare event in BTPE cannot be trusted.

Many applications use these implementations as part of the procedures, such as the efficient generation of random graphs from $\mathcal{G}(n,p)$ [3,19,4], logistic regression [9], generating virtual data sets in GLM analysis [1], generating random data or parameters [21] and speeding up Monte-Carlo simulation system [27].

⋆ This research was supported by NSF Grants IIS-1247750 and CCF-1114930.

Recently, both efficient and exact sampling algorithm for some distributions are developed, e.g. for normal and geometric distributions [15,5] but not yet for binomial distribution. In this paper, we present what is, to the best of our knowledge, the **first sublinear-time algorithm** for drawing a sample from a binomial distribution **with no loss of precision**. In particular, we show that:

Theorem 1. *Given a binomial distribution $B(n, p)$ for $n \in \mathbb{N}$, $p \in \mathbb{Q}$, drawing a sample from it takes $\mathcal{O}(\log^2 n)$ time using $\mathcal{O}(n^{1/2+\epsilon})$ space w.h.p., after $\mathcal{O}(n^{1/2+\epsilon})$-time preprocessing for small $\epsilon > 0$. The preprocessing does not depend on p and can be used for any $p' \in \mathbb{Q}$ and for any $n' \leq n$.*

Previous Work. Several sublinear-time algorithms have been described [17,7,14], although all of these trade precision for speed. Algorithms Binv [14] and Bg [6,14] both run in expected $\mathcal{O}(np)$ time. The former requires calculating $(1 - p)^n$; the latter requires calculating the ratio of two logarithms. Algorithm Balias [18,14] requires calculating $\binom{n}{k}$ for all k in $\{0, 1, \ldots, n\}$ and constructing an alias table [18] based on the calculated values. The alias table can be constructed in $\mathcal{O}(n)$ time and then each variate generation can be computed in $\mathcal{O}(1)$ time with bounded precision. Algorithm Btpe [14] divides the binomial distribution into parts and approximates each part by an upper-bound function. To pick a variate at random, the algorithm samples a variate following the distribution composed of the upper bound functions and accepts it with a probability equal to the ratio of the binomial distribution and upper bound function. The procedure is repeated if the test fails. This skill is known as the accept/reject rule, used in [17,14,7,27]. Btpe runs in sublinear time and is used by default in the statistical software R and GNU Scientific Library [22,10]. Because the distribution is divided piecewise and the piece is selected by an approximation to the true probability, Btpe does not exactly compute the binomial distribution. See [2,7,12,25,13] for more $\mathcal{O}(1)$-time algorithms in real computation model.

These algorithms run in sublinear time only if the precision of the calculations is truncated. When full precision is used, in calculating ratios, logarithms, or exponential functions, the time grows to at least linear. It is not clear how to modify them to be both accurate to full precision and sublinear.

Organization. In Section 2, preliminary definitions and building blocks are introduced. In Section 3, a simple algorithm is devised and further revised to achieve the claimed time complexity. Then, in Section 4, we conducted a set of experiments to compare the default algorithm used in GNU Scientific Library with the proposed one.

2 Preliminaries

To make it easier to understand the proposed algorithms, we sketch the outline of the algorithms and make definitions in this section. Given an input of a positive integer n and a real number $p \in [0, 1]$, the output is an integer $b \in \{0, 1, \ldots, n\}$ selected so that $\Pr[b = k] = \binom{n}{k} p^k (1 - p)^{n-k}$ for all $k \in \{0, 1, \ldots, n\}$; that is, a sample b is drawn from the binomial distribution $B(n,p)$ of n trials and success rate p. We analyze algorithms under the log-cost RAM model [20]. We assume that it takes $\mathcal{O}(1)$ time to do arithmetic calculations on constant number of operands of $\mathcal{O}(\log n)$ bits and to generate a fair binary random bit $u \in \{0,1\}$. We assume that p is rational and thus p can be represented by finite number of digits in base two, possibly repeating. Without loss of generality, let p be $(0.a_1 \cdots a_\ell \overline{a_{\ell+1} \cdots a_{\ell+r}})_2$. If $r > 0$, the part $a_{\ell+1} \cdots a_{\ell+r}$ repeats; otherwise, no part repeats. Formally, if $r > 0$, $a_i = a_{i-r}$ for all $i > \ell + r$; otherwise, $a_i = 0$ for all $i > \ell$.

Consider determining a Bernoulli trial T with success rate $p = (0.a_1 a_2 \cdots)_2$, as follows. Start by comparing a fairly-generated random bit u with a_1. If $u < a_1$, T returns a positive outcome; if $u > a_1$, T returns a negative outcome; otherwise $u = a_1$, proceed to the next digit and repeat the procedure. Then, expected $\mathcal{O}(1)$ comparisons are needed.

A binomial variate b can be sampled from $B(n,p)$ by simply checking how many of n Bernoulli trials have a positive outcome. We can mimic the single Bernoulli trial procedure above in which we replaced a comparison with p by a sequence of comparisons with a fair coin. The variate b is initialized to 0. Suppose we sample b_1 from $B(n,1/2)$. That means that b_1 trials had value 0 at the first sampled digit and the remaining $n-b_1$ trials had value 1. If $a_1 = 1$, then $b = b + b_1$, because all b_1 trials are less than p no matter what the remaining sampled digits are. Having determined the outcome of b_1 Bernoulli trials, set $n = n - b_1$. If $a_1 = 0$, then $n = b_1$, because $n - b_1$ trials are greater than p. We repeat this procedure until $n = 0$, which takes $\mathcal{O}(\log n)$ rounds, both in expectation and with high probability, considering that n will be roughly halved ($\leq n/2 + \sqrt{cn \ln n}$) in each round with probability $1 - \mathcal{O}(1/n^c)$.

In Section 3, we show how to construct a structure $S(n,c)$, a variation of *discrete distribution generating tree* [16]. Both construction time and used space are $\mathcal{O}((n \log^3 n)^{1/2})$. Affter which, sampling a variate from $B(n,1/2)$ takes $\mathcal{O}(\log n)$ time (matched the possible optimal bound [16]) with probability $1 - \mathcal{O}(1/n^c)$ for any constant $c \geq 1$. We also show how to construct $S(n,\infty)$ in $\mathcal{O}(n^2)$ time.

Note that drawing a variate from $B(n,p)$ by the above procedure possibly uses $B(n',1/2)$ for all $n' \in \{1,2,\ldots,n\}$. It would be too slow to construct $S(n',c)$ whenever we need to generate a variate from $B(n',1/2)$. A possible solution is to construct $S(n',c(n'))$ for all $n' \in \{1,2,4,\ldots,\lfloor\!\lfloor n \rfloor\!\rfloor\}$ where $c(n') = 4c$ if $n' > n^{1/4}$ or otherwise $c(n') = \infty$ and $\lfloor\!\lfloor x \rfloor\!\rfloor$ is the largest power of two no greater than x. To generate a variate b from $B(n',1/2)$ where n' is not a power of two, we decompose n' into $h = \mathcal{O}(\log n)$ powers of two $\omega_1, \omega_2, \ldots, \omega_h$, generate a variate b_i from each $B(\omega_i, 1/2)$, and let $b = \sum_i b_i$. Therefore, if we have the structures $S(n',c(n'))$ for all $n' \in \{1,2,4,\ldots,\lfloor\!\lfloor n \rfloor\!\rfloor\}$, generating a variate from $B(n',1/2)$ for any $n' \leq n$ takes $\mathcal{O}(\log^2 n)$ time with probability $1 - \Pr\left[\bigcup_i \mathcal{E}_i\right] \leq 1 - \sum_i \Pr\left[\mathcal{E}_i\right] = 1 - \mathcal{O}(h/n^c) = 1 - \mathcal{O}(\log n/n^c)$ where $\mathcal{E}_i$ denotes the event that generating b_i takes more than $\mathcal{O}(\log n)$ time. As a result, generating a variate from $B(n,p)$ takes $\mathcal{O}(\log^3 n)$ time with probability higher than $1 - \mathcal{O}(\log^2 n/n^c)$. We call this *simple sampling.*

Our *refined sampling* algorithm is based on the fact that every natural number is sum of four square numbers [26]. However, we are not going to construct $S(n', c(n'))$ for all $n' \in \mathcal{W} = \{k^2 : k \leq n\}$ because it would take too long. Instead, we construct a much smaller set $\mathcal{W}' \subset \mathcal{W}$ such that every natural number is the sum of h numbers in $\mathcal{W}'$, where h is a multiple of 4. Let $h = 16$, for example, $|\mathcal{W}'| \approx |\mathcal{W}|^{1/2}$. Therefore, whenever $S(n', c(n'))$ is needed and $n' \notin \mathcal{W}'$, we decompose n' into square numbers $w_1, w_2, \ldots, w_h \in \mathcal{W}'$, generate a variant b_i from each $B(w_i, 1/2)$, and let $b = \sum_i b_i$. Since h is constant, generating a variate from $B(n,p)$ takes $\mathcal{O}(\log^2 n)$ time with probability higher than $1 - \mathcal{O}(\log n/n^c)$.

3 Exact Binomial Sampling

In Section 2, we described how generating a variate from $B(n,p)$ can be reduced to generating variates from a sequence of $\mathcal{O}(\log n)$ $B(n', 1/2)$ with probability higher than $1 - \mathcal{O}(\log n/n^c)$ no matter what the precision of p is and where $n' \in \{1, 2, \ldots, n\}$. In this section, we begin by showing how to construct a structure $S(n,c)$ for generating a variate from $B(n, 1/2)$ in $\mathcal{O}(\log n)$ time with probability higher than $1 - \mathcal{O}(1/n^c)$. Then, we show that constructing $S(n', c)$ for each n' in a small subset of $\{1, 2, \ldots, n\}$ suffices to generate $B(n', 1/2)$ for each $n' \in \{1, 2, \ldots, n\}$.

In order to sample from $B(n, 1/2)$, we need to be careful of how many bits we use. To see why, consider that a generated variate b has value k with probability $P[b = k] = \binom{n}{k}/2^n$. The value of these probabilities can vary from 2^{-n} to $\Theta(1/\sqrt{n})$. Thus, if we are not careful, we end up manipulating probabilities that take n bits to represent.

The main idea will be to construct a structure that uses fewer bits to represent the needed probabilities. In most cases, this will be enough to correctly compute the variates. In order to compute the sample with exactly the correct probabilities, our structure $S(n,c)$ will change from time to time, but with low probability, expand the number of bits it uses to calculate the sample. Therefore, with high probability, $S(n,c)$ will be small and fast, but occasionally we might expand it.

Consider the event space for sampling. In order to generate a variate from $B(n, 1/2)$, one must pick an event e_i from $E = \{e_0, e_1, \ldots, e_n\}$ with probability $p_i = \binom{n}{i}/2^n$ and output the chosen i as the generated variate. We decompose each $p_i = \binom{n}{i}/2^n$ into ℓ_i powers of two which sum to p_i and replace each $e_i \in E$ with $e_{i1}, e_{i2}, \ldots, e_{i\ell_i}$. If e_{ij} is picked, then the generated variate is i. Each event e_{ij} from $\hat{E} = \{e_{ij} : 0 \leq i \leq n, 1 \leq j \leq \ell_i\}$ is selected with probability p_{ij}, each of which is a power of two. Note that there are many different ways to decompose a probability p_i into powers of two, though in only one of them does each power of two occur only once. We will consider decompositions in which some powers of two might appear twice so that it leaves the flexibility of computing p_i incrementally without worrying about carries in arithmetic operations. The distribution on the p_{ij} is heavily biased towards a few high-probability events, which correspond to the most significant bits of the binomial coefficients in base two.

We construct a *power tree* on $\hat{E}$ in order to draw an event e_{ij} from $\hat{E}$ with probability p_{ij}. Let a power tree be a binary tree in which each node of depth d is associated with a probability 2^{-d}. The set of probabilities on the leaf nodes of $T_{\hat{E}}$ are exactly $\hat{P} = \{p_{ij} : e_{ij} \in \hat{E}\}$, in one-to-one correspondence, in an arbitrary order. A variate can be generated according to $\hat{P}$ by taking a random, fair root-to-leaf path in $T_{\hat{E}}$.

To construct a power tree, we start with a root node associated with probability 1 and make it an unlabelled leaf. We process each $e_{ij} \in \hat{E}$ in an arbitrary order as follows. When processing e_{ij}, find a candidate unlabeled leaf with smallest probability p no smaller than p_{ij}. If $p = p_{ij}$, label the leaf with e_{ij}. Otherwise, replace the candidate leaf with two children, each with half the probability, and make one leaf the new candidate. Proceed until a leaf has been labelled with e_{ij}.

Note that after processing each e_{ij}, no two unlabelled leaves have the same probability. Therefore, if $Q \subseteq \hat{E}$ is the set of events we have processed so far and $p_Q = \min\{p_{ij} : e_{ij} \in Q\}$, there are $|Q|$ labelled leaves and at most $1 + \log(1/p_Q)$ unlabelled leaves, so the tree has size $\mathcal{O}(|Q| + \log(1/p_Q))$ in total.

Lemma 1. *Given an event set $\hat{E}$, the power tree on $\hat{E}$ can be constructed in $\mathcal{O}(|\hat{E}|)$ time using $\mathcal{O}(|\hat{E}|)$ space. The construction is incremental, adding one event at a time. During the construction, if some $Q \subseteq \hat{E}$ has processed, both running time and used space are $\mathcal{O}(|Q| + \log(1/p_Q))$.*

Proof. We maintain an array of pointers, $\rho_0, \rho_1, \dots, \rho_{\log(1/p_Q)}$, where ρ_i points to the unique unlabelled leaf with probability 2^{-i}, if such a leaf exists. Then, it takes $\mathcal{O}(1)$ time to process an event e_{ij} if $\rho_{\log(1/p_{ij})}$ points to an unlabelled node; otherwise, we have to find the unlabelled leaf with smallest p no smaller than p_{ij} by traversing the pointer array in $\mathcal{O}(\log(1/p_{ij}) - \log(1/p))$ time. Simple charging scheme, in which the event that consumes a leaf is charged for creating the leaf, yields the bound.

We show that the construction never fails as follows. Let U be the set of probabilities of unlabelled leaves. For each coming e_{ij}, the associated probability $p_{ij} \leq \sum_{p \in U} p$. If the procedure fails as e_{ij} comes, then it means we cannot find $p \in U, p \geq p_{ij}$. Since each $p \in U$ is an unique power of two, if each $p \in U, p < p_{ij}$, then $\sum_{p \in U} p < p_{ij}$, a contradiction. □

The set $\hat{E}$ is as large as $\mathcal{O}(n^2)$ for the power tree of $B(n, 1/2)$ because each p_i requires $O(n)$ bits, as noted above. Therefore, fully constructing power tree for $B(n, 1/2)$, i.e. $S(n, \infty)$, takes $\mathcal{O}(n^2)$ time. We will construct the power tree of a set Q chosen so that it has $o(n)$ events and its power tree has depth $O(\log n)$, but so that the total probability of $\hat{E} - Q$ is polynomially small. Lemma 2 establishes that such a set always exists. If we sample using the power tree of Q, we will almost always reach a labelled leaf. If we reach an unlabelled leaf, we complete the construct of the power tree of $\hat{E}$ to finish the sampling procedure. That is, once a random walk reaches an unlabeled leaf, we postpone the on-going random walk, complete the unfinished part of the power tree construction, and then resume the random walk at the point where it is postponed. Thus, with very high probability, we will use small space and time to sample from $\hat{E}$.

Since the size of Q depends on how we decompose p_i of $e_i \in \hat{E}$, we describe the decomposition of p_i first to complete the claim that Q has small size and then describe the motivation of p_i's decomposition. For each p_i, we find two positive numbers $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ which add to p_i. The first is roughly the high-order $(c+1)\log n$ bits of p_i and the second is roughly the remaining (low order) bits. Ideally, they would be exactly the high and lower order bits, but it would be computationally expensive to find the high-order bits of $\binom{n}{i}$, considering the difficulty to determine certain bit of the product of two integers [23]. As the computation efficiency is concerned, to make $\mathcal{H}_s(p_i)$ to be of short length and $\mathcal{L}_s(p_i)$ to be of small value, we let $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ have an overlap as follows.

The value $\mathcal{L}_s(p_i) = p_i - \mathcal{H}_s(p_i)$, so we only need to worry about $\mathcal{H}_s(p_i)$. We defer the discussion of exactly how to compute $\mathcal{H}_s(p_i)$ until later and here note that $\mathcal{H}_s(p_i) = k \cdot 2^s$ for non-negative integer k and $p_i - \mathcal{H}_s(p_i) < 2 \cdot 2^s$. When we drop the subscript s, the default value of $s = -(c+1)\log n$. Note that, if $\mathcal{H}_r(p_i)$ for $r < s$ is given, one can get $\mathcal{H}_s(p_i)$ by truncating the trailing $s - r$ bits of $\mathcal{H}_r(p_i)$ because $\mathcal{L}_r(p_i) < 2 \cdot 2^r \leq 2^s$ and trailing $s - r$ bits of $\mathcal{H}_r(p_i)$ are less than 2^s, which add up to a value less than $2 \cdot 2^s$. We will show that such an $\mathcal{H}_s(p_i)$ can always be selected. And the only probability that a power of two potentially gets repeated in such a decomposition of $\mathcal{H}_s(p_i)$ and $\mathcal{L}_s(p_i)$ is 2^s.

The high order bits of many events are all zeros because the binomial coefficient are strongly concentrated. We define the major range where the events of non-zero high order bits located to be $\mathcal{C}(n) = [n/2 \pm (cn \ln n)^{1/2}]$ (C for center). Trivially, the number of bits is $(c+1)\log n * 2(cn \ln n)^{1/2}$, so picking this for Q is small. Lemma 2 shows that the probability is high. Because every $\mathcal{H}(p_i)$ is a multiple of $2^{-(c+1)\log n}$, this bounds the depth of the power tree to be $\mathcal{O}(c \log n)$.

To obtain Q, we decompose $\mathcal{H}(p_i)$ into its binary representation for $i \in \mathcal{C}(n)$. The following lemma completes the claim that $|Q|$ is small. Let bit(p) be the set of powers of two in the binary representation of p.

Lemma 2. *Let $\mathcal{P}(E)$ be the sum of probabilities associated with the events in E. Let Q be the set of events associated with the probability in*

$$\bigcup_{i \in \mathcal{C}(n)} \mathrm{bit}(\mathcal{H}(p_i)). \tag{2}$$

Then, $|Q|$ is $\mathcal{O}((c^3 n \log^3 n)^{1/2})$ and $\mathcal{P}(Q) \geq 1 - \mathcal{O}(1/n^c)$.

Proof. Let Q_1 and Q_2 be the set of events associated with the probability in, respectively,

$$\bigcup_{i \notin \mathcal{C}(n)} \mathrm{bit}(\mathcal{H}(p_i)) \cup \mathrm{bit}(\mathcal{L}(p_i)) \text{ and } \bigcup_{i \in [0,n]} \mathrm{bit}(\mathcal{L}(p_i)). \tag{3}$$

We have $\mathcal{P}(Q_1) \leq 2/n^c$ by Chernoff bound [8] and $\mathcal{P}(Q_2) \leq 2(n+1)/n^{c+1}$ by the definition of $\mathcal{L}(p_i)$. Because $\overline{Q} = Q_1 \cup Q_2$,

$$\mathcal{P}(Q) \geq 1 - \mathcal{P}(Q_1) - \mathcal{P}(Q_2) = 1 - \mathcal{O}(1/n^c) \tag{4}$$

as desired. Then, $|Q| = (cn \log n)^{1/2}(c+1)\log n = \mathcal{O}((c^3 n \log^3 n)^{1/2})$. □

We claim that $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$ can be efficiently computed. We first show how to compute $\mathcal{H}(p_i)$ for $i \in \mathcal{C}(n)$ from $\mathcal{H}(p_{\lfloor n/2 \rfloor})$ and then show how to compute $\mathcal{H}(p_{\lfloor n/2 \rfloor})$ itself. Given the $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s = (c+2)\log n$, Lemma 3 shows how we can compute $\mathcal{H}(p_i)$ for $i \in \mathcal{C}(n)$ in $\mathcal{O}((c^3 n \log n)^{1/2})$ time. In other words, if we have the central probability of the binomial computed to more precision, we can use that to compute the surrounding values.

Lemma 3. *Given $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s \leq -(c+2)\log n$, it takes $\mathcal{O}((c^3 n \log n)^{1/2})$ time to calculate $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$.*

Proof. Let $\lfloor a \rfloor_s \equiv \lfloor a/2^s \rfloor 2^s$. Given $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$, we claim that $\mathcal{H}(p_{\lfloor n/2 \rfloor - 1})$ can be obtained by truncating some trailing bits in $\ell_1 \equiv \lfloor \mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 \rfloor_s$ where $r_1 = \binom{n}{\lfloor n/2 \rfloor - 1} / \binom{n}{\lfloor n/2 \rfloor} = \lfloor n/2 \rfloor / (\lfloor n/2 \rfloor + 1) < 1$ because

$$\begin{aligned}
p_{\lfloor n/2 \rfloor - 1} - \ell_1 &= p_{\lfloor n/2 \rfloor} * r_1 - \ell_1 \\
&= (\mathcal{H}_s(p_{\lfloor n/2 \rfloor}) + \mathcal{L}_s(p_{\lfloor n/2 \rfloor})) * r_1 - \ell_1 \\
&= (\mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 - \lfloor \mathcal{H}_s(p_{\lfloor n/2 \rfloor}) * r_1 \rfloor_s) + \mathcal{L}_s(p_{\lfloor n/2 \rfloor}) * r_1 \\
&< 2^s + 2 \cdot 2^s < 3 \cdot 2^s.
\end{aligned}$$

Then, ℓ_1 has $-s$ bits and $p_{\lfloor n/2 \rfloor - 1} - \ell_1 < 3 \cdot 2^s$. If we get ℓ_1' by truncating the last two bits in ℓ_1, then ℓ_1' has $-s-2$ bits and $p_{\lfloor n/2 \rfloor - 1} - \ell_1' < 3 \cdot 2^s + 3 \cdot 2^s < 2 \cdot 2^{s+2}$. Hence, ℓ_1' is a valid $\mathcal{H}_{s+2}(p_{\lfloor n/2 \rfloor - 1})$ and we have $\mathcal{H}(p_{\lfloor n/2 \rfloor - 1})$ if $s \leq -(c+2)\log n$. Similarly, we claim that $\mathcal{H}(p_{\lfloor n/2 \rfloor - 2})$ can be obtained by truncating some trailing bits in $\ell_2 \equiv \lfloor \ell_1 * r_2 \rfloor_s$ where $r_2 = (\lfloor n/2 \rfloor - 1)/(\lfloor n/2 \rfloor + 2) < 1$ because

$$\begin{aligned}
p_{\lfloor n/2 \rfloor - 2} - \ell_2 &= p_{\lfloor n/2 \rfloor - 1} * r_2 - \ell_2 \\
&= (\ell_1 + (p_{\lfloor n/2 \rfloor - 1} - \ell_1)) * r_2 - \ell_2 \\
&= (\ell_1 * r_2 - \lfloor \ell_1 * r_2 \rfloor_s) + (p_{\lfloor n/2 \rfloor - 1} - \ell_1) \\
&< 2^s + 3 \cdot 2^s < 4 \cdot 2^s
\end{aligned}$$

Again, we get ℓ_2' by truncating the last two bits in ℓ_2, then ℓ_2' has $-s-2$ bits and $p_{\lfloor n/2 \rfloor - 2} - \ell_2' < 3 \cdot 2^s + 4 \cdot 2^s < 2 \cdot 2^{s+2}$ as desired. Clearly, $p_{\lfloor n/2 \rfloor - k} - \ell_k < (k+2) \cdot 2^s$ and we can calculate $\mathcal{H}(p_i)$ for all $i \in \mathcal{C}(n)$ if $s = -(c+2)\log n$. Since each ℓ_i' can be calculated by $\mathcal{O}(1)$ arithmetic calculations on operands of $\mathcal{O}(c \log n)$ bits, we need $\mathcal{O}(c)$ time for each ℓ_i' and thus $\mathcal{O}((c^3 n \log n)^{1/2})$ in total. □

To compute $\mathcal{H}_s(p_{\lfloor n/2 \rfloor})$ for $s = -(c+2)\log n$, we exploit the idea of computing Q; that is, consider summing up the terms of $i \in \mathcal{C}(n)$ in the equality

$$\binom{2k}{k} = \sum_i \binom{k}{i}^2 \approx \sum_{i \in \mathcal{C}(n)} \mathcal{H}\left(\binom{k}{i}\right)^2. \tag{5}$$

Suppose $\binom{k}{\lfloor k/2 \rfloor}$ has been computed, we use it to compute $\binom{k}{i}$ for $i \in \mathcal{C}(k)$ in $\mathcal{O}((c^3 k \log k)^{1/2})$ time by Lemma 3 and use all of them to compute $\binom{2k}{k}$ by Equation 5. To compute $\mathcal{H}_s(\binom{2k}{k})$ with $s = 2k - (c+2)\log 2k$ (i.e. $\mathcal{H}_s(\binom{2k}{k}/2^{2k})$ with

$s = -(c+2)\log 2k$), we need $\mathcal{H}_s\left(\binom{k}{\lfloor k/2\rfloor}\right)$ with $s = k-(c+4+1/2)\log k$ implied by Lemma 4 if $k \geq 2^{(c+7)/2}$. Otherwise $k < 2^{(c+7)/2}$, the binomial coefficient $\binom{k}{\lfloor k/2\rfloor}$ can be computed in $\mathcal{O}(1)$ time because c is a constant. Consequently, for even n, the computation of $\binom{n}{n/2}$ can be reduced to the computation of $\binom{n/2}{\lfloor n/2\rfloor}$; for odd n, the computation of $\binom{n}{\lfloor n/2\rfloor}$ could simply add $\binom{n-1}{\lfloor n/2\rfloor}$ and $\binom{n-1}{\lfloor n/2\rfloor-1}$, each of which is a case of even n. We apply this procedure to compute $\binom{n}{\lfloor n/2\rfloor}$ recursively and the time complexity is

$$\sum_{k=0}^{\log n} \mathcal{O}(((c+5k/2+2)^3(n/2^k)\log(n/2^k))^{1/2}) = \mathcal{O}((n\log^2 n)^{1/2}), \tag{6}$$

dominated by the construction time of the power tree, remarked in Theorem 2.

Lemma 4. *Let* $p_k = \mathcal{H}_s^2\left(\binom{n}{k}\right)/\binom{2n}{n}$ *with* $s \leq n-(2c+1/2)\log n$. *Then,*

$$\sum_{k\in\mathcal{C}(n)} p_k \geq 1 - 18/n^{2c}.$$

Proof.

$$\sum_{k\in[0,n]} \binom{n}{k}^2 - \sum_{k\in\mathcal{C}(n)} \mathcal{H}_s^2\left(\binom{n}{k}\right) = \sum_{k\notin\mathcal{C}(n)} \binom{n}{k}^2 \tag{7}$$

$$+ \sum_{k\in\mathcal{C}(n)} 2\binom{n}{k}\mathcal{L}_s\left(\binom{n}{k}\right) - \mathcal{L}_s^2\left(\binom{n}{k}\right) \tag{8}$$

Consider the sum of sampled n values from a pool of n 0's and 1's without replacement. The probability of the sum being k is $q_k = \binom{n}{k}^2/\binom{2n}{n}$. By Corollary 1.1 in [24], we have $\sum_{k\notin\mathcal{C}(n)} q_k \leq 2\exp[-4(c\ln n)/(1+1/n)] \leq 2/n^{2c}$. Then, $(7)/\binom{2n}{n} \leq 2/n^{2c}$.

By definition, $\mathcal{L}_s(\binom{n}{k}) < 2\cdot 2^s$ and therefore $(8) \leq 4\cdot 2^s\cdot 2^n = 2^{n+s+2}$. Because $\binom{2n}{n} \geq 2^{2n-2}/n^{1/2}$ by Stirling's approximation, $(8)/\binom{2n}{n} \leq 16/n^{2c}$. Putting the results together, we have

$$\sum_{k\in\mathcal{C}(n)} \mathcal{H}_s^2\left(\binom{n}{k}\right)/\binom{2n}{n} \geq 1 - 18/n^{2c}. \qquad \square$$

Theorem 2. *To compute* $S(n,c)$, *it takes* $\mathcal{O}((n\log^3 n)^{1/2})$ *time, after which, it is stored in* $\mathcal{O}((n\log^3 n)^{1/2})$ *space. Given* $S(n,c)$, *it takes* $\mathcal{O}(\log n)$ *time to generate a variate from* $B(n,1/2)$ *with probability* $1-\mathcal{O}(1/n^c)$.

Now we have $S(n,c)$ for generating variates from $B(n,1/2)$ and show how to use it for $B(n,p)$ as follows; that is, building a set of $S(n',c(n'))$. We suppress the term $c(n')$ for convience without changing the time complexity.

Simple Sampling. generates a variate from $B(n,p)$ for $n \in \mathbb{N}, p \in \mathbb{Q}$ denoted by $(0.a_1a_2\cdots)_2$ using the structure $S(n)$ for all $n' \in \{1,2,4,\ldots,\llbracket n \rrbracket\}$.

In Section 2, we have shown how to use $B(n',1/2)$ for $n' \in [n]$ to generate a variate from $B(n,p)$. Because $S(n')$ might not be contained in the constructed structure $\{S(1),S(2),S(4),\ldots,S(n)\}$, to generate a variate from $B(n',1/2)$, we generate a variate from each of $B(\omega_1,1/2),\ldots,B(\omega_h,1/2)$ and add the variates, where ω_i are powers of two added to n'. Clearly, h can be $\mathcal{O}(\log n')$.

There are $\mathcal{O}(\log n)$ steps in the reduction from $B(n,p)$ to $B(n',1/2)$ for $n' \in [n]$. Each step requires to generate a variate from $B(n',1/2)$, where n' can be decomposed into $\mathcal{O}(\log n')$ powers of two. Thus, it takes $\mathcal{O}(\log^2 n)$ time to generate a variate from $B(n',1/2)$ with probability $1-\mathcal{O}(\log n/n^c)$ by union bound. Considering that the number of steps is $\mathcal{O}(\log n)$ with probability $1-\mathcal{O}(1/n^c)$, the total running time for generating $B(n,p)$ is therefore $\mathcal{O}(\log^3 n)$ with probability $1-\mathcal{O}(\log^2 n/n^c)$.

Refined Sampling. is as the simple sampling but selects an integer set R such that every positive integer $n' \leq n$ is a sum of j elements in R, where j is a constant. In this way, to generate a variate from $S(n')$, one can generate a variate from each of $S(w_1),S(w_2),\ldots,S(w_j)$ and add the generated variates, where $n' = w_1+w_2+\cdots+w_j$. $R' = \{k^2 \leq n : k \in \mathbb{N}\}$ is a possible candidate for R because each positive integer is a sum of at most four square numbers [26]. However, we do not build the data structure $S(n')$ for all $n' \in R'$ because the time complexity $\sum_{k\in\{1,4,9,\ldots,n\}} \mathcal{O}((n\log^3 n)^{1/2}) = \mathcal{O}((n^2\log^3 n)^{1/2})$ is too much. A better candidate for R could be

$$R_1 \cup R_2 = \{k^2 \leq n : k \in \mathbb{N}, k^2 \equiv 0 \bmod t\} \cup \{k^2 \leq n : k \in \mathbb{N}, k^2 < t\}, \qquad (9)$$

where t is a chosen square number. Let n' be represented as w_1t+w_2. Because w_1t (resp. w_2) is a sum of at most 4 square numbers in R_1 (resp. R_2), every integer $n' \leq n$ is a sum of at most 4×2 numbers in $R_1 \cup R_2$. Thus, the time complexity is reduced to $\mathcal{O}((t^2\log^3 t)^{1/2}) + \mathcal{O}((t(n/t)^2\log^3 n)^{1/2})$ or $\mathcal{O}((n^{4/3}\log^3 n)^{1/2})$ by letting $t = n^{2/3}$. Similarly, let $n' = w_1t_1+w_2t_2+\cdots+w_h$, the time complexity can be furtherly reduced to $\mathcal{O}((n^{2^h/(2^h-1)}\log^3 n)^{1/2}) = \mathcal{O}(n^{1/2+\epsilon})$ for some constant h and small $\epsilon > 0$. To decompose each w_it_i into four square numbers in the corresponding set R_i in $\mathcal{O}(1)$ time, we preprocess a small table for lookup. The small table is used to decompose integers no more than $n^{1/8+\epsilon}$ into four square numbers, whose construction time is bounded by $\mathcal{O}((n^{1/8+\epsilon})^4)$ dominated by the main procedure. Therefore, there is no problem to decompose $w_1,w_2,\ldots,w_{h-2}$ because all of them are no more than $n^{1/8+\epsilon}$. For w_{h-1} bounded by $n^{1/4+\epsilon}$, one can first decompose w_{h-1} as $x+y$, where x is the largest square number smaller than w_{h-1} and thus $y = \mathcal{O}(n^{1/8+\epsilon})$ so that it can be decomposed by table-lookup in $\mathcal{O}(1)$ time. Similar decomposition is applied to w_h. As a result, the mentioned constant $j = 4h+3$.

As the arguments used for the simple sampling, generating a variate from $B(n,p)$ takes $\mathcal{O}(\log^2 n)$ time with probability $1-\mathcal{O}(\log n/n^c)$ after $\mathcal{O}(n^{1/2+\epsilon})$-time preprocessing.

4 Empirical Evaluation

In this section, we conducted experiments to compare the quality of generated variates and to compare the computation time used for generating variates among the algorithms with and without loss of precision. We compare our proposed algorithm with Btpe [14], which is the default algorithm for binomial sampling in both R [22] and GNU Scientific Library (GSL) [10]. The implementation of GSL is used to conduct the experiments.

We generated 10^8 random variates from $B(2^{30}, 1/2)$ and plotted a histogram of outputs. Btpe demonstrated a discontinuity, as shown in Figure 1.

Our algorithm takes poly-logarithmic time. In contrast, Btpe takes approximately constant time. As shown in Figure 2, the computation time is about the same for $n \leq 2^{15}$ and increases to 3.5 times more for $n = 2^{30}$. When we stored the power tree in pre-order, the running time of our algorithm improved, suggesting that cache misses were to blame. Therefore, our algorithm could benefit from more tuning.

We implemented our algorithm in C++ with GNU Scientific Library [10] and GNU Multiple Precision Arithmetic Library [11], compiled it with G++4.63 with optimization flag -O3. The machine we used is equipped with a Celeron G530 2.4GHz CPU and 2GB of 1066MHz RAM. The operating system is Ubuntu 12.04 Desktop. The computation time is measured by wall time, i.e., the elapsed time.

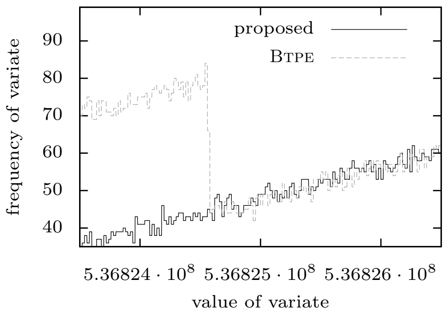


Fig. 1. The histograph of results for Btpe in discontinuous $B(2^{30}, 1/2)$. The histograph is smoothed by window averaging, with a window of size 20. Btpe oversamples the tail 0.74% of the time, that is, by a factor of 2.59.

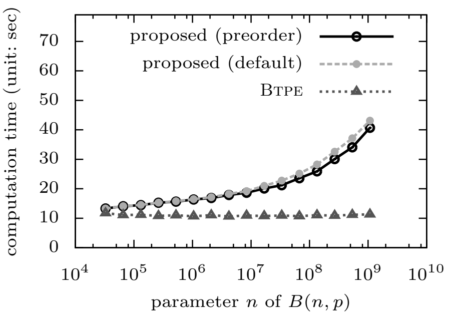


Fig. 2. Compare the computation time used for generating 10^8 variates from $B(n, 1/2)$ by different algorithms. Each data point is an average of 10 experiments.

References

1. Abe, J., Kamimura, Y.: Do female parasitoid wasps recognize and adjust sex ratios to build cooperative relationships? Journal of Evolutionary Biology 25(7), 1427–1437 (2012)
2. Ahrens, J.H., Dieter, U.: Sampling from binomial and poisson distributions: A method with bounded computation times. Computing 25(3), 193–208 (1980)
3. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E 71(3), 36113 (2005)

4. Blanca, A., Mihail, M.: Efficient generation - close to G(n,p) and generalizations. CoRR abs/1204.5834 (2012)
5. Bringmann, K., Friedrich, T.: Exact and efficient generation of geometric random variates and random graphs. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 267–278. Springer, Heidelberg (2013)
6. Devroye, L.: Generating the maximum of independent identically distributed random variables. Computers and Mathematics with Applications 6(3), 305–315 (1980)
7. Devroye, L.: Non-Uniform Random Variate Generation. Springer (1986)
8. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley (January 1968)
9. Fox, J., Weisberg, S.: An R Companion to Applied Regression. SAGE Publications (2010)
10. Galassi, M., et al.: Gnu Scientific Library: Reference Manual. Network Theory Ltd. (2003)
11. Granlund, T.: The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, 5.0.5 edn. (2012), `http://gmplib.org/`
12. Hörmann, W.: The generation of binomial random variates. Journal of Statistical Computation and Simulation 46, 101–110 (1993)
13. Hörmann, W., Leydold, J., Derflinger, G.: Automatic nonuniform random variate generation. Springer (2004)
14. Kachitvichyanukul, V., Schmeiser, B.W.: Binomial random variate generation. Commun. ACM 31, 216–222 (1988)
15. Karney, C.F.F.: Sampling exactly from the normal distribution. CoRR abs/1303.6257 (2013)
16. Knuth, D., Yao, A.: The complexity of nonuniform random number generation. In: Algorithms and Complexity: New Directions and Recent Results. Academic Press (1976)
17. Knuth, D.E.: The art of computer programming. Seminumerical algorithms, vol. 2. Addison-Wesley Longman Publishing Co., Inc. (1997)
18. Kronmal, R.A., Peterson, A.V.J.: On the alias method for generating random variables from a discrete distribution. The American Statistician 33(4), 214–218 (1979)
19. Miller, J.C., Hagberg, A.: Efficient generation of networks with given expected degrees. In: Frieze, A., Horn, P., Prałat, P. (eds.) WAW 2011. LNCS, vol. 6732, pp. 115–126. Springer, Heidelberg (2011)
20. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press (1995)
21. Patterson, R.S.: of Louisville, U.: Testing the Effects of Predictors Using Data Generated by Non-identity Link Functions of the Single-index Model: A Monte Carlo Approach. University of Louisville (2008)
22. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing (2008)
23. Sauerhoff, M., Woelfel, P.: Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In: STOC, pp. 186–195. ACM (2003)
24. Serfling, R.J.: Probability Inequalities for the Sum in Sampling without Replacement. The Annals of Statistics 2(1), 39–48 (1974)
25. Stadlober, E., Zechner, H.: The patchwork rejection technique for sampling from unimodal distributions. ACM Trans. Model. Comput. Simul. 9(1), 59–80 (1999)
26. Stillwell, J.: Elements of Number Theory. Springer (2003)
27. Tsai, M.-T., Wang, D.-W., Liau, C.-J., Hsu, T.-S.: Heterogeneous subset sampling. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 500–509. Springer, Heidelberg (2010)

Trivial, Tractable, Hard. A Not So Sudden Complexity Jump in Neighborhood Restricted CNF Formulas

Dominik Scheder[⋆,⋆⋆,⋆⋆⋆]

Aarhus University

Abstract. For a CNF formula F we define its 1-conflict graph as follows: Two clauses $C, D \in F$ are connected by an edge if they have a nontrivial resolvent – that is, if there is a unique literal $u \in C$ for which $\bar{u} \in D$. Let $\mathrm{lc}_1(F)$ denote the maximum degree of this graph.

A k-CNF formula is a CNF formula in which each clause has exactly k distinct literals. We show that (1) a k-CNF formula F with $\mathrm{lc}_1(F) \leq k-1$ is satisfiable; (2) there are unsatisfiable k-CNF formulas F with $\mathrm{lc}_1(F) = k$; (3) there is a polynomial time algorithm deciding whether a k-CNF formula F with $\mathrm{lc}_1(F) = k$ is satisfiable; (4) satisfiability of k-CNF formulas F with $\mathrm{lc}_1(F) \leq k+1$ is NP-hard.

Furthermore, we show that if F is a k-CNF formula and $\mathrm{lc}_1(F) \leq k$, then we can find in polynomial time a satisfying assignment (if F is satisfiable) or a treelike resolution refutation with at most $|F|$ leaves (if F is unsatisfiable). Here, $|F|$ is the number of clauses of F.

1 Introduction

There are several parameters to measure the structural complexity of CNF formulas, and they influence the computational complexity of their associated satisfiability decision problem. Some of them yield a fixed-parameter tractable problem – for example the treewidth of formulas (Allender, Chen, Lou, Papakonstantinou, and Tang [1]). For other parameters we are hit by the full power of NP-completeness once the parameter is large enough. Think of k, the maximum clause width of a formula: For $k = 2$ we know polynomial algorithms, for $k \geq 3$ the problem is NP-complete. In this paper we define in a natural way a graph on the clauses of the formula and investigate the complexity of the satisfiability problem depending on the maximum degree of this graph. We connect two

⋆ The author acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, and from the CFEM research center (supported by the Danish Strategic Research Council), within which this work was performed.

⋆⋆ Research was supported by the SNF Grant 200021-118001/1.

⋆⋆⋆ The author acknowledges support from the Simons Institute for the Theory of Computing, UC Berkeley.

clauses C, D of our formula with an edge if those clauses have a non-trivial resolvent. That is, if there is exactly one literal $u \in C$ for which $\bar{u} \in D$. We call this the 1-conflict graph. Thus, the degree of a clause C in this graph is the number of potential resolution partners in the formula. This graph is similar to the one defined by Ostrowski, Grégoire, Mazure, and Sais [2].[1] We show that a k-CNF formula is satisfiable if its 1-conflict graph has maximum degree at most $k-1$; the satisfiability problem is NP-hard if we allow a maximum degree of $k+1$; in between, for k-CNF formulas graphs of maximum degree k, there is a nontrivial algorithm that runs in polynomial time. If the formula is satisfiable, the algorithm returns a satisfying assignment. If it is unsatisfiable, it returns a treelike resolution refutation of size at most $2m-1$, where m is the number of clauses.

1.1 Notions of Degree, Neighborhood, and Conflict

A k-CNF formula in which every variable appears in at most $2^k/(ek)$ clauses is satisfiable. This is a direct consequence of the Lovász Local Lemma [3] and was first observed by Kratochvíl, Savický, and Tuza [4]. There is no reason to believe that $2^k/(ek)$ is tight. This motivates the following definition: Let $f(k)$ be the largest integer d such that every k-CNF formula F with $\Delta(F) \leq d$ is satisfiable. Here, Δ is the "maximum degree" of a formula: The maximum number of clauses in which a variable appears. The above result shows that $f(k) \geq 2^k/(ek)$. Proving matching upper bounds, i.e., constructing unsatisfiable k-CNF formulas of low maximum variable degree, turned out to be not trivial at all. The upper bound has been improved in several papers, to $O\left(2^k/k^{0.26}\right)$ by Savický and Sgall [5] and to $O\left(2^k \log k/k\right)$ by Hoory and Szeider [6]. Gebauer [7] improved it to $O\left(2^k/k\right)$, which is tight up to a constant factor, and finally Gebauer, Szabó, Tardos [8] proved that $f(k) = (1 \pm o(1))2^{k+1}/ek$, i.e., they even found the right constant factor.

How difficult is satisfiability of k-CNF formulas of bounded degree? Let (k,d)-SAT denote the problem of deciding whether a given k-CNF formula F of maximum degree $\Delta(F) \leq d$ is satisfiable. Clearly, $(k, f(k))$-SAT is trivial: All instances are satisfiable. Kratochvíl, Savický, and Tuza [4] showed that (k,d)-SAT exhibits a complexity jump: When the number of permitted occurrences per variable increases from $f(k)$ to $f(k)+1$, the complexity of the decision problem jumps from trivial (all instances are satisfiable) to NP-complete. It seems surprising that one can prove such a result without knowing the value of $f(k)$.

Other structural parameters exhibit complexity jumps, too. For a clause C in a CNF formula F, let $\Gamma_F(C)$ denote the clauses of F (excluding C) that have at least one variable in common with C, regardless of its sign. Let $\Gamma(F) := \max_{C\in F} |\Gamma_F(C)|$. Again by the Lovász Local Lemma, every k-CNF formula F with $\Gamma(F) \leq 2^k/e - 1$ is satisfiable. Gebauer, Moser, Welzl, and myself [9]

[1] Their graph has edges also between clauses with a trivial resolvent, for example $(x \vee y \vee \bar{z}), (u \vee \bar{y} \vee z)$, which is labeled as a trivial edge. Our graph is such the subgraph of all non-trivial edges of the graph of Ostrowski et al.

showed that there is some number $\ell(k)$ such that (1) all k-CNF formulas F with $\Gamma(F) \leq \ell(k)$ are satisfiable; (2) there exists an unsatisfiable k-CNF formula F with $\Gamma(F) \leq \ell(k) + 1$; (3) satisfiability of k-CNF formulas F with $\Gamma(F) \geq \max(k + 3, \ell(k) + 2)$ is NP-hard. Note that $k + 3 \leq \ell(k) + 2$ for sufficiently large k. This means an "almost sudden" complexity jump, where in the case $\Gamma(F) = \ell(k) + 1$ the decision problem is neither known to be in P nor to be NP-complete.

Define $\Gamma'_F(C)$ to be the number of clauses in F with which C has a *conflict*, that is those clauses D for which $u \in C$ and $\bar{u} \in D$ for some literal u. Let $\mathrm{lc}(F) := \max_{C \in F} |\Gamma'_F(C)|$. Here, lc stands for *local conflict*. The *lopsided* Lovász Local Lemma shows that every k-CNF formula F with $\mathrm{lc}(F) \leq 2^k/(ek) - 1$ is satisfiable. In [9] it was proven that this notion of conflict degree exhibits a sudden complexity jump: There is a function $\mathrm{lc}(k)$ such that (1) all k-CNF formulas F with $\mathrm{lc}(F) \leq \mathrm{lc}(k)$ are satisfiable; (2) deciding satisfiability of k-CNF formulas F with $\mathrm{lc}(F) \geq \mathrm{lc}(k) + 1$ is NP-hard.

1.2 Our Contribution

Two clauses C, D have a *1-conflict* if there is exactly one literal u such that $u \in C$ and $\bar{u} \in D$. In other words, if C and D have a non-trivial resolvent. For example, the clauses $\{x, y, z\}$ and $\{\bar{x}, y\}$ have a 1-conflict, but $\{x, y, z\}$ and $\{\bar{x}, \bar{z}\}$ do not. We denote by $\Gamma^1_F(C)$ the set of clauses $D \in F$ such that C and D have a 1-conflict, and $\mathrm{lc}_1(F) := \max_{C \in F} |\Gamma^1_F(C)|$. In contrast to $\Delta(F)$, $\Gamma(F)$ and $\mathrm{lc}(F)$, it turns out that we completely understand the complexity of k-SAT when we restrict lc_1:

Theorem 1 (Complexity Jump). *The following three statements hold for all $k \geq 0$:*

1. *Every k-CNF formula F with $\mathrm{lc}_1(F) \leq k - 1$ is satisfiable.*
2. *There exists an unsatisfiable k-CNF formula F with $\mathrm{lc}_1(F) = k$.*
3. *Satisfiability of k-CNF formulas F with $\mathrm{lc}_1(F) \leq k$ is in P.*
4. *Deciding satisfiability of k-CNF formulas F with $\mathrm{lc}_1(F) \leq k + 1$ is NP-complete, if $k \geq 3$.*

Let us say a word about the proof of this theorem. Point 2 is very simple, we just provide a construction of a k-CNF formula for every $k \in \mathbb{N}$. Point 4, the hardness result, uses a reduction that is very similar to that of Kratochvíl, Savický, and Tuza [4] and Gebauer, Moser, Welzl, and myself [9]. Point 1 uses the concept of *blocked clauses* (Kullmann [10]). These are special clauses that are redundant and can be removed. The proof of Point 3 is the most interesting in our opinion. It consists of two main observations: (1) It is enough to decide satisfiability separately for each connected component of the 1-conflict graphs. (2) If the 1-conflict graph is connected, then splitting on a variable and iteratively deleting blocked clauses drastically reduces the size of the input formula. Blocked clause elimination (Järvisalo, Biere, and Heule [11]; Ostrowski, Grégoire, Mazure, and Sais [2]) is a known preprocessing step in SAT solvers and is quite useful in practice. In theory, however, eliminating blocked clauses can increase the resolution complexity of a formula exponentially: There are examples of formulas with short resolution proofs, but if one removes blocked clauses, every resolution proof of the remaining formula most be of exponential size; see for example Cook [12]. The class of formulas we discuss in Point 3 is thus *not* of this form: Blocked clause elimination is provably beneficial here.

Point 3 shows that there is a provable gap between the trivial and the NP-hard regime of the parameter lc_1. Such a gap is non-existent or not known to exist for the other parameters discussed above. We give a SAT algorithm that is correct in general, and in the special case of k-CNF formulas F with $\mathrm{lc}_1(F) \leq k$ runs in polynomial time. It is a branching algorithm and thus produces a treelike resolution refutation whose size is bounded by the number of recursive calls (this is a well-known fact; for a proof see [13], Theorem 3.2.5, page 35). Therefore, we get the following theorem:

Theorem 2 (Short Resolution Proofs). *If F is an unsatisfiable k-CNF formula and $\mathrm{lc}_1(F) = k$, then there is a treelike resolution refutation of F with at most $|F|$ leaves, where $|F|$ is the number of clauses in F.*

Theorem 3 (Finding the Satisfying Assignment). *Suppose F is a satisfiable k-CNF formula and $\mathrm{lc}_1(F) \leq k$. Then we can find a satisfying assignment in polynomial time.*

2 Notation

A CNF formula is a conjunction (AND) of clauses: $C_1 \wedge \cdots \wedge C_m$. A clause is a disjunction (OR) of literals: $x \vee \bar{y} \vee z$, where a literal is either a variable or its negation. We typically let n denote the number of variables in a formula, m the number of clauses, and k the size of its clauses: In a k-CNF formula, all clauses have size k. For notational purposes, we view formulas as set of clauses and clauses as sets of literals. So $\{\{x, y\}, \{\bar{x}, \bar{y}\}\}$ is the 2-CNF formula $(x \vee y) \wedge (\bar{x} \vee \bar{y})$ (which by the way is equivalent to $x \oplus y$). By $\mathrm{vbl}(C)$ and $\mathrm{vbl}(F)$ we denote the set of variables in a clause C or formula F, respectively. For a clause $D = \{u_1, \ldots, u_k\}$, we write $\bar{D} := \{\bar{u}_1, \ldots, \bar{u}_k\}$. This is not the negation of D. For a CNF formula F and a variable x, $F^{[x \mapsto 1]}$ is the CNF formula we obtain by replacing x by the constant 1. Thus, every clause containing x is satisfied (and can be removed from F), and every occurrence of $\bar{x}$ is unsatisfied and can be removed. We define $F^{[x \mapsto 0]}$ analogously.

2.1 Resolution

If C and D have a one-conflict, i.e., $C \cap \bar{D} = \{u\}$, we call the clause $E := (C \setminus \{u\}) \cup (D \setminus \bar{u})$ the resolvent of C and D. It is an easy exercise to show that the formulas $C \wedge D$ and $C \wedge D \wedge E$ are equivalent. Let F be a CNF formula. A *resolution derivation* from F is a sequence of clauses $C_1, C_2, \ldots, C_m$ where each C_i is (1) a clause of F or (2) the resolvent of two earlier clauses in the sequence. It is not difficult to see that F implies each clause in the sequence; that is, any assignment satisfying F satisfies $C_1, \ldots, C_m$. If $C_m = \Box$, i.e., the empty clause, which always evaluates to 0, we call $C_1, \ldots, C_m$ a resolution *refutation*, as it shows that F is unsatisfiable. A *treelike resolution derivation* from F is a binary tree T with the following properties: Every vertex u is labeled with a clause C_u; a leaf is labeled with a clause of F; if an inner vertex u has children v and w, then C_u is the resolvent of C_v and C_w. If the root is labeled with the empty clause $\Box$, we call it a treelike resolution refutation of F.

3 Proofs

We prove Point 2 of Theorem 1, which is the simplest of the four points. Take k variables and let F_k be the k-CNF formula containing all 2^k k-clauses over the k variables. F_k is unsatisfiable and $\mathrm{lc}_1(F_k) = k$. For an alternative construction, take $2k - 1$ variables and let G_k consist of all $\binom{2k-1}{k}$ completely positive k-clauses and all $\binom{2k-1}{k}$ completely negative k-clauses. Again one checks that G_k is unsatisfiable and $\mathrm{lc}_1(G_k) = k$. For example, for $k = 2$ those two constructions yield

$$\{\{x, y\}, \{\bar{x}, y\}, \{x, \bar{y}\}, \{\bar{x}, \bar{y}\}\} \tag{1}$$

and

$$\{\{x, y\}, \{x, z\}, \{y, z\}, \{\bar{x}, \bar{y}\}, \{\bar{x}, \bar{z}\}, \{\bar{y}, \bar{z}\}\} \ . \tag{2}$$

Their 1-conflict graphs are a C_4 and a C_6, respectively.

3.1 Basic Properties of the 1-Conflict Graph

Before we attack the remaining three points of the theorem, let us collect some interesting facts about 1-conflicts. Let us start with a simple but surprising observation, which probably is folklore.

Proposition 1. *Every CNF formula F with $\Box \notin F$ and $\mathrm{lc}_1(F) = 0$ is satisfiable.*

Note that without that proposition, the notion "1-conflict" would be misleading. After all, under any reasonable notion of conflict, a formula without conflicts should be satisfiable (extreme cases like $\Box \in F$ excluded). A direct consequence of the above proposition is that a hypergraph in which $|e \cap f| \neq 1$ for all hyperedges e, f is 2-colorable. This is a result of Lovász (Problem 13.33 in [14]).

Proof. A CNF formula F is unsatisfiable if and only if there is a resolution derivation of the empty clause (For a proof use induction over the number of variables or see for example [15], Theorem 4.2.1, page 26). Since F has no 1-conflicts, we cannot build any new resolvents. Since $\Box \notin F$, the formula is satisfiable. $\Box$

Lemma 1. *A CNF formula F is satisfiable if and only if every connect component of its 1-conflict graph is satisfiable. Furthermore, given satisfying assignments $\alpha_1, \ldots, \alpha_t$ for each of its t connected components, we can efficiently find a satisfying assignment α of F.*

Again, this is something we expect from a reasonable notion of conflict.

Proof. One direction is trivial: If F is satisfiable, then all connected components are satisfiable. For the other direction, write $F = F_1 \uplus F_2$ such that there is no 1-conflict between F_1 and F_2. By induction on the number of connected components, both F_1 and F_2 are satisfiable.

Choose a pair α_1, α_2 of assignments to $\text{vbl}(F)$ such that α_1 satisfies F_1, α_2 satisfies F_2, and the Hamming distance $d_H(\alpha_1, \alpha_2)$ is minimized. We claim that α_1 satisfies F_2 as well, and therefore F. Suppose for the sake of contradiction that this is not the case. There is a clause $D \in F_2$ such that α_1 does not satisfy D. Since α_2 satisfies D, there is a literal $u \in D$ such that $\alpha_1(u) = 0$ and $\alpha_2(u) = 1$. Define $\alpha_1' := \alpha[u \mapsto 1]$. Clearly $d_H(\alpha_1', \alpha_2) = d_H(\alpha_1, \alpha_2) - 1$. If we can prove that α_1' still satisfies F_1, we have arrived at a contradiction to d_H being minimal, and are done. Consider any $C \in F_1$. By the assumptions of the lemma, there is no 1-conflict between C and D. Hence either $C \cap \bar{D} = \emptyset$ or $|C \cap \bar{D}| \geq 2$. In the first case, $\alpha_1(C) = \alpha_1'(C) = 1$. In the second case, α_1 satisfies at least two literals in C, and therefore, α_1' satisfies at least one literal in C. This shows that α_1' indeed satisfies F_1, contradicting minimality of $d_H(\alpha_1, \alpha_2)$.

As for the algorithmic aspect, suppose we are given assignments α_1 and α_2 satisfying F_1 and F_2, respectively. As above, we we locally modify α_1, reducing the Hamming distance between to α_2, until we arrive at a single assignment α satisfying both F_1 and F_2. This takes only polynomial time. □

3.2 Blocked Literals and Blocked Clauses

It will pay off to introduce some notation. Let F be a CNF formula, C a clause, and $u \in C$ a literal. Define $\Gamma_F^1(C, u) := \{D \in F \mid C \cap \bar{D} = \{u\}\}$, that is, those clauses that have a 1-conflict with C, and this 1-conflict is generated by u. Note that

$$\Gamma_F^1(C) = \bigcup_{u \in C} \Gamma_F^1(C, u) \ ,$$

and this union is a disjoint one.

Definition 1 (Blocking Literals and Blocked Clauses, Kullmann [10]). *We say u* blocks *C* in *F if $\Gamma_F^1(C, u) = \emptyset$. A clause C is* blocked *in F if some $u \in C$ blocks C in F.*

If the ambient formula is understood, we simply say that u blocks C and C is blocked, not explicitly referring to F. Blocked clauses are redundant, in some way:

Proposition 2 (Kullmann [10]). *Let F be a CNF formula and $C \in F$ some clause. If C is blocked in F, then F is satisfiable if and only if $F \setminus \{C\}$ is. Furthermore, given a satisfying assignment α of $F \setminus \{C\}$, we can efficiently find a satisfying assignment α' of F.*

We can use Proposition 2 to repeatedly remove blocked clauses in a formula, finally arriving at a formula without blocked clauses, which we denote by $\texttt{deleteBlocked}(F)$.

Proposition 3 (Kullmann [10]). *Let F be a CNF formula and let $F' :=$ $\texttt{deleteBlocked}(F)$. Then F is satisfiable if and only if F' is, and given a satisfying assignment α' of F', we can efficiently construct a satisfying assignment α of F.*

Proof. This follows from Proposition 2 and induction on the number of clauses. $\square$

Proposition 3 yields another proof of Proposition 1: If $\mathrm{lc}_1(F) = 0$, then every non-empty clause is blocked by one of its literals. The algorithm `deleteBlocked` will remove one by one, finally arriving at the empty formula, which is satisfiable. Here we were using an innocent but crucial fact: If a clause C is blocked with respect to F, then it is also blocked with respect to every subformula $F' \subseteq F$ for which $C \in F'$.

This proves Point 1 of the theorem: If F is a k-CNF formula and $\mathrm{lc}_1(F) \leq k-1$, then every clause contains at least one literal that blocks it. Thus $\texttt{deleteBlocked}(F) = \{\}$, the empty formula, thus it is satisfiable.

3.3 Simple and Tight Formulas

Definition 2. *A CNF formula F is* simple *if $|\Gamma_F(C, u)| \leq 1$ for every $C \in F$ and every $u \in C$. It is* tight *if $|\Gamma_F(C, u)| = 1$ for every $C \in F$ and every $u \in C$.*

For example, the following formula is tight and satisfiable.

$$\{\{\bar{x}_1, x_2\}, \{\bar{x}_2, x_3\}, \ldots, \{\bar{x}_{n-1}, x_n\}, \{\bar{x}_n, x_1\}\}$$

As another example, the formulas in (1) and (2) are tight and unsatisfiable.

Proposition 4. *Suppose F is simple. Then $\texttt{deleteBlocked}(F)$ is tight. Suppose F is a k-CNF formula and $\mathrm{lc}_1(F) \leq k$. Then $\texttt{deleteBlocked}(F)$ is tight.*

Proof. Suppose F is simple. Then any subformula $F' \subseteq F$ is simple, too. Thus $F' := \texttt{deleteBlocked}(F) \subseteq F$ is simple. It contains no blocked clauses, so $|\Gamma_{F'}(C, u)| \geq 1$ for all $u \in C \in F'$. But $|\Gamma_{F'}(C, u)| \leq |\Gamma_F(C, u)| \leq 1$, which means they must be exactly 1. In other words, F' is tight.

For the second statement, suppose F is a k-CNF formula and $\mathrm{lc}_1(F) \leq k$. Then this statement is true for $F' := \texttt{deleteBlocked}(F)$, too. Since F' contains no blocked clause, $|\Gamma_{F'}(C, u)| \geq 1$ for all $u \in C \in F'$. Thus $k \leq \sum_{u \in C} |\Gamma_{F'}(C, u)| = |\Gamma_{F'}(C)| \leq \mathrm{lc}_1(F') = k$, so equality holds throughout, meaning $|\Gamma_{F'}(C, u)| = 1$, and F' is tight. $\square$

Proposition 5. *Suppose F is simple, and x is a variable. Then $F^{[x\mapsto 1]}$ is simple, and so is $F^{[x\mapsto 0]}$.*

Proof. Suppose $F' := F^{[x\mapsto 0]}$ is not simple. We will show that F is not simple. By assumption on F', there is a clause $C' \in F'$ and a literal $u \in C'$ such that $|\Gamma_{F'}(C', u)| \geq 2$. This means there are clauses D'_1, D'_2 such that $C' \cap \bar{D}'_1 = C' \cap \bar{D}'_2 = \{u\}$. Since $F' = F^{[x\mapsto 0]}$, this means that F contains clauses C, D_1, D_2 such that either $C = C'$ or $C = C' \vee x$; either $D_1 = D'_1$ or $D_1 = D' \vee x$; either $D_2 = D'_2$ or $D_2 = D' \vee x$. None of those clauses contains $\bar{x}$, though. Therefore $C \cap \bar{D}_1 = C' \cap \bar{D}'_1 = \{u\}$, and similarly $C \cap \bar{D}_2 = C' \cap \bar{D}_2 = \{u\}$. Thus, $D_1, D_2 \in \Gamma^1_F(C, u)$, and F is not simple, either. $\square$

3.4 An Efficient Algorithm

We will now use the above notions of blocked clauses and simple and tight formulas to prove the main result of this paper, i.e., Point 3 of Theorem 1. We give an algorithm that efficiently decides satisfiability of k-CNF formulas F with $\mathrm{lc}_1(F) \leq k$. See Algorithm `simpleSAT` below. To see the correctness `simpleSAT`,

Algorithm 1.1. `simpleSAT`(CNF formula F)

```
1: $F \leftarrow$ deleteBlocked($F$)
2: if $\square \in F$ then
3:     return false
4: else if $F = \{\}$ then
5:     return true
6: else if $F = F_1 \uplus F_2$ for some $F_1, F_2 \neq \{\}$ and $|C \cap \bar{D}| \neq 1$ for all $C \in F_1, D \in F_2$
   then
7:     return simpleSAT($F_1$) $\wedge$ simpleSAT($F_2$)
8: else
9:     $x \leftarrow \mathrm{vbl}(F)$
10:    $G_1 :=$ deleteBlocked($F^{[x\mapsto 1]}$)
11:    $G_0 :=$ deleteBlocked($F^{[x\mapsto 0]}$)
12:    return simpleSAT($G_1$) $\vee$ simpleSAT($G_0$)
13: end if
```

consider lines 1.1 and 1.1. The algorithm recurses on F_1 and F_2 and returns `true` if both calls return `true`. By Lemma 1, F is satisfiable if and only if F_1 and F_2 are both satisfiable individually. The challenging part is to argue that its running time is polynomial in our case.

Lemma 2. *If F is simple, then* `simpleSAT`(F) *runs in polynomial time. More precisely, let m be the number of clauses in F. The total number of calls to* `simpleSAT`(F) *during its execution is $2m - 1$ if $m \geq 1$ and 1 otherwise.*

Proof. If $m = 0$, then $F = \{\}$ and the algorithm just returns `true`. So the claim holds for $m = 0$. After the first line, F is tight, which follows from Proposition 4. If $m = 1$, then $F = \{\Box\}$ or $F = \{\}$ after the first line, so there is no further recursive call either. So the claim holds for $m = 1$, too.

Otherwise, suppose $m \geq 2$, i.e., F has at least two clauses. Then `simpleSAT` either recurses on two subformulas F_1, F_2 (line 1.1) or on G_0, G_1 (line 1.1). Suppose `simpleSAT` recurses on F_1 and F_2. Note that both F_1 and F_2 have at least one clause. We apply induction to F_1 and F_2 and see that the total number of calls is at most $1 + (2|F_1| - 1) + (2|F_2| - 1) = 2(|F_1| + |F_2|) - 1 = 2m - 1$. If `simpleSAT` recurses on G_0 and G_1, things are more complicated. This is the only point where we need that F is tight:

Proposition 6. *Suppose F is tight, $x \in \text{vbl}(F)$, and let $G_0 :=$ `deleteBlocked` $(F^{[x \mapsto 0]})$ and $G_1 :=$ `deleteBlocked`$(F^{[x \mapsto 1]})$. Then $|G_0| + |G_1| \leq |F|$.*

With this proposition, we apply induction to G_0 and G_1. If both G_0 and G_1 contain at least one clause, then the total number of calls is $1 + (2|G_0| - 1) + (2|G_1| - 1) \leq 2|F| - 1$.

At this point we are almost done, but have to deal with the annoying special case that G_0 or G_1 might be empty. If G_0 contains no clause but G_1 does, then we apply induction on G_1 and see that the number of calls is $1+1+(2|G_1|-1) = 2|G_1| + 1 \leq 2|F| - 1$, since $|G_1| < |F|$. If $G_0 = G_1 = \{\}$, then there is a total of 3 calls. Since F has $m \geq 2$ clauses, this completes the proof of the lemma. $\Box$.

Proof (of Proposition 6). Let $C \in F$ be a clause. We argue that C may make its way into either G_0 or G_1, but not both. Thus $|G_0| + |G_1| \leq F$.

There are two cases: Suppose $x \in C$ or $\bar{x} \in C$, without loss of generality $x \in C$. Then setting $x \mapsto 1$ satisfies C, and C does not make it into G_1, but $C^{[x \mapsto 0]}$ may make it into G_0 (provided it survives `deleteBlocked`).

So suppose $x \notin C$ and $\bar{x} \notin C$. After line 1.1, the 1-conflict graph of F is connected. So there is a path $C = C_1, C_2, \ldots, C_{t-1}, C_t$ such that $x \in \text{vbl}(C_t)$ but $x \notin \text{vbl}(C_i)$ for $1 \leq i \leq t - 1$. Without loss of generality, $x \in C_t$. After setting $x \mapsto 1$, the clause C_t is satisfied, and C_{t-1} has one 1-conflict neighbor less. So now C_{t-1} is blocked, and `deleteBlocked`$(F^{[x \mapsto 1]})$ deletes it. Thus C_{t-2} loses a neighbor and becomes blocked, and so on, until finally $C = C_1$ will be removed. See Figure 1 for an illustration. Thus, C does not make its way into G_1. This proves the proposition. $\Box$

Resolution Size – Proof of Theorem 2

The algorithm `simpleSAT` is a branching algorithm, and it is a well-known fact that branching algorithms implicitly produce a treelike resolution refutation when run on an unsatisfiable formula (see e.g. [13], Theorem 3.2.5, page 35). The number of clauses in the refutation is at most the number of recursive calls. Since an unsatisfiable formula has $m \geq 1$ clauses, the number of calls is at most $2m - 1$, by Lemma 2. Thus the resolution tree has at most $2m - 1$ nodes, and therefore at most m leaves. This proves Theorem 2.

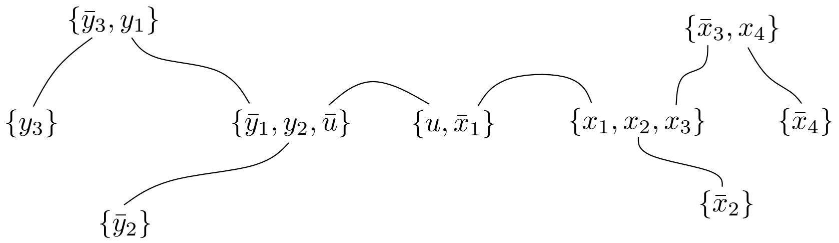


Fig. 1. Illustration of Proposition 6. When we set u to 1, the clause $\{u, \bar{x}_1\}$ disappears. The clause $\{x_1, x_2, x_3\}$ has only one outgoing edge labeled x_1. Once $\{u, \bar{x}_1\}$ disappears, the literal x_1 will block $\{x_1, x_2, x_3\}$, and $\{x_1, x_2, x_3\}$ will be deleted, too. Then $\bar{x}_2$ will block $\{\bar{x}_2\}$ and $\bar{x}_3$ will block $\{\bar{x}_3, x_4\}$, thus these clauses are also deleted, and so on.

Finding the Satisfying Assignment – Proof of Theorem 3

Suppose F is a satisfiable k-CNF formula and $\mathrm{lc}_1(F) \leq k$. We construct a satisfying assignment F by tracking the execution of `simpleSAT`(F). Denote by $F' :=$ `deleteBlocked`(F) the input formula after the first line. By Proposition 4, F' is tight. If we can efficiently find a satisfying of F', then by Proposition 3 we can efficiently find a satisfying assignment of F. If `simpleSAT` recurses in line 1.1 on F_1 and F_2, we assume by induction that we know satisfying assignments α_1 of F_1 and α_2 of F_2. By Lemma 1 we can efficiently combine α_1, α_2 into a single α satisfying of F'. If `simpleSAT` recurses in line 1.1 on G_0 and G_1, suppose without loss of generality that G_1 is satisfiable and let α be a satisfying assignment. Since $G_1 =$ `deleteBlocked`$(F'^{[x \mapsto 1]})$, we can efficiently find a satisfying assignment α' of $F'^{[x \mapsto 1]}$, by Proposition 3. Thus, $\alpha' \cup [x \mapsto 1]$ satisfies F'. Summing up, we can construct a satisfying assignment of F by adding some bookkeeping to `simpleSAT`.

4 Hardness for $\mathrm{lc}_1 \geq k + 1$: Proof Sketch

We sketch a reduction from k-SAT to k-SAT with $\mathrm{lc}(F) \leq k + 1$, but refer the reader to the appendix for the full details. Let F be a k-CNF formula and let $\deg_F(x)$ denote the number of clauses in F in which x occurs, regardless of its sign. In a first step, we introduce $2\deg_F(x)$ new variables $x_1, x_2, \ldots, x_{2\deg_F(x)}$ for each $x \in \mathrm{vbl}(F)$ and replace the i^{th} occurrence of x by x_{2i}.

In a second step, we add an *equalizer formula* $\mathrm{Eq}(x_1, \ldots, x_{2\deg_F(x)})$ for each $x \in \mathrm{vbl}(F)$. This is a 2-CNF formula which is satisfied if and only if its $2\deg_F(x)$ variables receive the same truth value. The resulting formula is satisfiable if and only if F is. However, it is not a k-CNF formula, because it contains 2-clauses.

In a third step, we "fill up" each 2-clause $\{u, v\}$ to a k-clause, by adding $k - 2$ new variables. This is, we replace $\{u, v\}$ by $\{u, v, w_3, \ldots, w_k\}$. Finally, we add a "forcer" for every new variable w_i introduced in the third step. A forcer is a k-CNF formula that is satisfiable, but only if w_i is set to 0. Such a forcer can be built in a rather straightforward manner from an unsatisfiable k-CNF formula G with $\mathrm{lc}_1(G) = k$.

Acknowledgments. I am very thankful to Heidi Gebauer, Robin Moser, and Emo Welzl. The question posed and answered in this paper arose during our work on [9]. Furthermore, I am grateful to Navid Talebanfard, who pointed out Exercise 13.33 from [14].

References

1. Allender, E., Chen, S., Lou, T., Papakonstantinou, P., Tang, B.: Width-parameterized SAT: time-space tradeoffs. ECCC TR12-027 (2012)
2. Ostrowski, R., Grégoire, É., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 185–199. Springer, Heidelberg (2002)
3. Erdős, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Hajnal, A., Rado, R., Sós, V.T. (eds.) Infinite and Finite Sets (to Paul Erdős on his 60th birthday), vol. II, pp. 609–627. North-Holland (1975)
4. Kratochvíl, J., Savický, P., Tuza, Z.: One more occurrence of variables makes satisfiability jump from trivial to NP-complete. SIAM Journal of Computing 22(1), 203–210 (1993)
5. Savický, P., Sgall, J.: DNF tautologies with a limited number of occurrences of every variable. Theoret. Comput. Sci. 238(1-2), 495–498 (2000)
6. Hoory, S., Szeider, S.: A note on unsatisfiable k-CNF formulas with few occurrences per variable. SIAM Journal on Discrete Mathematics 20(2), 523–528 (2006)
7. Gebauer, H.: Disproof of the neighborhood conjecture with implications to SAT. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 764–775. Springer, Heidelberg (2009)
8. Gebauer, H., Szabó, T., Tardos, G.: The local lemma is tight for SAT. In: Randall, D. (ed.) SODA, pp. 664–674. SIAM (2011)
9. Gebauer, H., Moser, R.A., Scheder, D., Welzl, E.: The Lovász Local Lemma and satisfiability. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 30–54. Springer, Heidelberg (2009)
10. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96-97, 149–176 (1999)
11. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
12. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. SIGACT News 8(4), 28–32 (1976)
13. Hoffmann, J.: Resolution proofs and DLL algorithms with clause learning. Diploma Thesis, Ludwig-Maximilians-Universität München (2007)
14. Lovász, L.: Combinatorial problems and exercises, 2nd edn. North-Holland (1993)
15. Krajicek, J.: Bounded Arithmetic, Propositional Logic and Complexity Theory. Encyclopedia of Mathematics and its Applications. Cambridge University Press (1995)

Dynamic Point Labeling is Strongly PSPACE-Complete

Kevin Buchin and Dirk H.P. Gerrits

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, the Netherlands

Abstract. An important but strongly NP-hard problem in automated cartography is how to best place textual labels for point features on a static map. We examine the complexity of various generalizations of this problem for dynamic and/or interactive maps. Specifically, we show that it is strongly PSPACE-complete to decide whether there is a smooth dynamic labeling (function from time to static labelings) when the points move, when points are added and removed, or when the user pans, rotates, and/or zooms their view of the points.

1 Introduction

Map labeling involves associating textual *labels* with certain *features* on a map such as cities (points), roads (polylines), and lakes (polygons). This task takes considerable time to do manually, and for some applications *cannot* be done manually beforehand. In air traffic control, for example, a set of moving points (airplanes) has to be labeled at all times. In interactive maps users may pan, rotate, and/or zoom their view of the map, which may also require relabeling. It is therefore unsurprising that map labeling has attracted considerable algorithmic research (see, for instance, the on-line Map Labeling Bibliography [9], currently containing 371 references).

Static Points. A good labeling for a point set has legible labels, and an unambiguous association between the labels and the points. This has been formalized by regarding the labels as axis-aligned rectangles slightly larger than the text

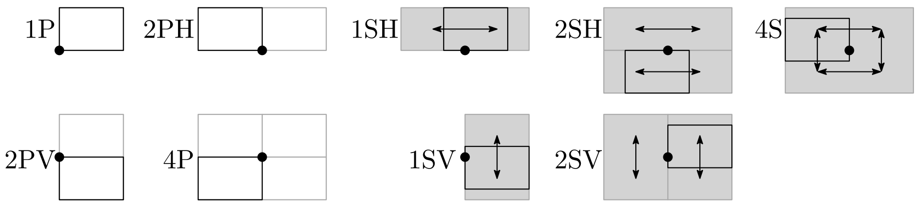


Fig. 1. A label model specifies the allowed positions (shown in gray) for the label of a point. Fixed-position models: the 1-position (1P), 2-position (2PH, 2PV), and 4-position (4P) models. Slider models: the 1-slider (1SH, 1SV), 2-slider (2SH, 2SV), and 4-slider (4S) models.

they contain, which must be placed without overlap so that each contains the point it labels on its boundary. Not all placements are equally desirable, and as such various *label models* have been proposed which specify allowed positions for the labels (Fig. 1). In the *fixed-position models*, every point has a finite number of label candidates. In particular, in the 1-, 2-, and 4-position models a subset of 1, 2, or 4 corners is designated and each label must have one of these coincide with the point it labels. The *slider models* generalize this. In the 1-slider models one side of each label is designated, but the label may contain its point anywhere on this side. In the 2-slider models there is a choice between two opposite sides, and in the 4-slider model the label can contain the point anywhere on its boundary.

Ideally, one would label all points with non-intersecting labels, but this is not always possible. Deciding this is a strongly NP-complete problem for the 4-position [4] and 4-slider [7] models. We may deal with this difficulty in several ways. Firstly, we may shrink the labels. The *size-maximization problem* asks to label all points with non-intersecting labels of maximal size. Secondly, we may remove labels. The *number-maximization problem* asks to label a maximum-cardinality subset of the points with non-intersecting labels of given dimensions. Thirdly, we may allow labels to overlap, but try to keep such occurrences to a minimum. The *free-label-maximization problem* asks for all points to be labeled with labels of given dimensions, maximizing the number of non-intersecting labels. As the decision problem mentioned above is strongly NP-hard for the 4-position and 4-slider models, these three optimization problems are as well.

Dynamic Points. A natural generalization of static point labeling is dynamic point labeling. Here the point set P changes over time, by points being added and removed, and/or by points moving continuously. This can be inherent to the point set (as in air traffic control), or be the result of the user panning, rotating, and zooming (as in interactive maps). Thus, the input is a *dynamic point set* P, which specifies for each point $p \in P$ its arrival and departure times, as well as its trajectory. We seek a *dynamic labeling* $\mathcal{L}$, which for all t assigns a static labeling $\mathcal{L}(t)$ to the static points $P(t)$ present at time t. For the 4-slider model we require that labels move continuously. For the fixed-position and 2-slider models we must allow labels to make "jumps". We only allow p's label to jump from position A to position B, if there is no candidate position C in between A and B in clockwise (or counter-clockwise) order around p. For the 4-position model we thus allow horizontal and vertical jumps, but no diagonal ones. For a 2-slider model we only allow jumps from an endpoint of one slider to the "same" endpoint of the other slider.

For static point labeling, practical heuristic algorithms and theoretical algorithms with guaranteed approximation ratios abound. Dynamic point labeling, however, has seen very few theoretical results. Been et al. [1] studied number-maximization for points under zooming, giving constant-factor approximations for unit-square labels in the 1-position model. Gemsa et al. [5] gave a PTAS for the same problem with rotation instead of zooming. Theoretical treatment of other label models and general point trajectories are sorely missing.

Our Results. We believe the relative lack of theoretical results for dynamic point labeling is not due to a lack of attempts. Intuitively, dynamic point labeling should be much harder than its static counterpart. We prove and quantify this intuition. Specifically, we consider the problem of deciding whether there exists a dynamic labeling without intersections for a given dynamic point set. We prove that this is PSPACE-complete for unit-square labels in the 4-position, 2-slider, and 4-slider models, and remains so even if the input is given in unary notation (it is *strongly* PSPACE-complete). This is the case when points are added or removed from the point set, when (some of) the points move, and when the point set is panned, rotated, or zoomed within a finite viewport. Any dynamic generalization of the mentioned static optimization problems is therefore strongly PSPACE-hard in these settings. Additionally, we prove that label-size maximization on dynamic point sets admits no PTAS unless P=PSPACE.

2 Structure of the Reduction

To prove PSPACE-hardness of dynamic point labeling, we reduce from *non-deterministic constraint logic (NCL)* [6], which is an abstract, single-player game. The game board is a *constraint graph*: an undirected graph with non-negative weights on both the vertices and the edges. A *configuration* of the constraint graph specifies an orientation for each of its edges. A configuration is *legal* if and only if each vertex's *inflow* (the summed weight of its incoming edges) is at least its own weight. The outflow of vertices, on the other hand, is not constrained. To make a *move* in this game is to reverse a single edge in a legal configuration such that the resulting configuration is again legal. Hearn and Demaine [6] showed that each of the following questions is PSPACE-complete.

- *Configuration-to-configuration NCL*: Given two legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, is there a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$?
- *Configuration-to-edge NCL*: Given a legal configuration $\mathcal{C}_A$ and an orientation for a single edge e_B, is there a sequence of moves transforming $\mathcal{C}_A$ into a legal configuration $\mathcal{C}_B$ in which e_B has the specified orientation?
- *Edge-to-edge NCL*: Given orientations for edges e_A and e_B, do there exist legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, and a sequence of moves transforming the one into the other, such that e_A has the specified orientation in $\mathcal{C}_A$ and e_B has the specified orientation in $\mathcal{C}_B$?

These decision problems remain PSPACE-complete even for planar, 3-regular constraint graphs consisting only of AND vertices and protected OR vertices. An *AND vertex* has a weight of 2, and its three incident edges have weights 1, 1, 2. To orient the weight-2 edge away from the vertex requires both weight-1 edges to be oriented towards the vertex. An *OR vertex* also has weight 2, as do its three incident edges. Thus at least one edge needs to be oriented towards the vertex at all times. An OR vertex is called *protected* if it has two edges that, because of constraints imposed on them by the rest of the constraint graph, cannot both be directed inward.

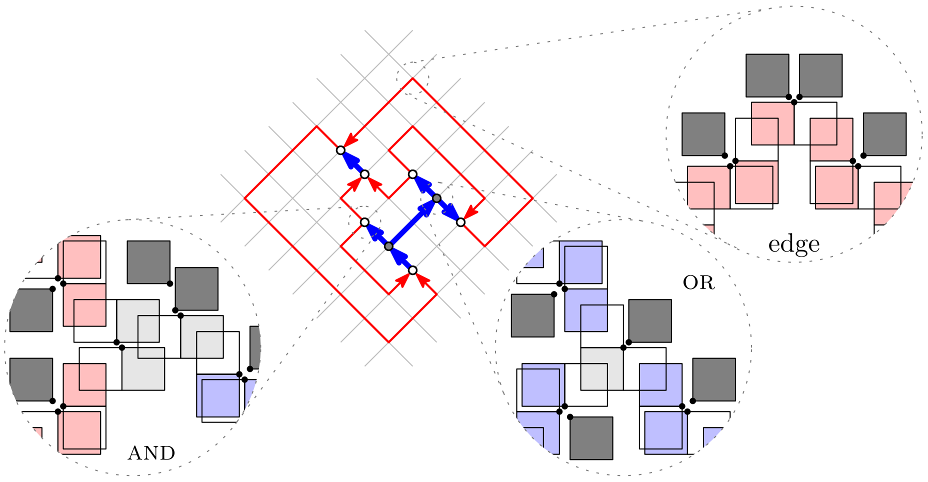


Fig. 2. Our construction for simulating non-deterministic constraint logic with dynamic point labeling. In this figure, and all that follow, weight-1 edges are drawn as thin, red lines and weight-2 edges are drawn as thick, blue lines.

As a first step towards proving hardness of dynamic labeling, we will show how to simulate a constraint graph G with a point set P. This will be done in such a way that labelings of P will correspond to configurations of G, and label movements will correspond to changing the orientations of edges in G. In the next section, we will then show how to use this construction to prove the PSPACE-hardness of various dynamic labeling problems.

At a high level, our construction works as follows. Given a planar constraint graph G with n vertices, we first embed it on a regular grid that has been turned by 45° relative to the coordinate axes. In this embedding G's vertices lie on grid vertices, and its edges form interior-disjoint paths along grid lines, as depicted in the center of Fig. 2. From the structure of Hearn and Demaine's hardness proof of NCL [6] we may in fact assume that such an embedding of G is given, but otherwise we can compute one in $O(n)$ time, for example with an algorithm of Biedl and Kant [3]. The next step is to replace each vertex and edge by appropriate *gadgets* built out of points that are to receive labels. In the figure, our gadgets are depicted in the circular insets. We call the points that have been depicted with dark gray labels *blockers*, as we can consider their labels to be fixed obstacles: labeling them differently than shown will only restrict the placement of other labels further. As is, the depicted blockers restrict the red and blue labels to two possible positions each in the 4-position model. These correspond to the two different orientations of edges. Labels "directed" *into* a vertex gadget (such as the red labels in the depicted AND gadget) correspond to edges pointing *out* of the vertex. Conversely, labels directed *out* of a vertex gadget (such as the blue labels in the depicted AND gadget) correspond to edges pointing *into* the vertex. The inflow constraints are then enforced by the light

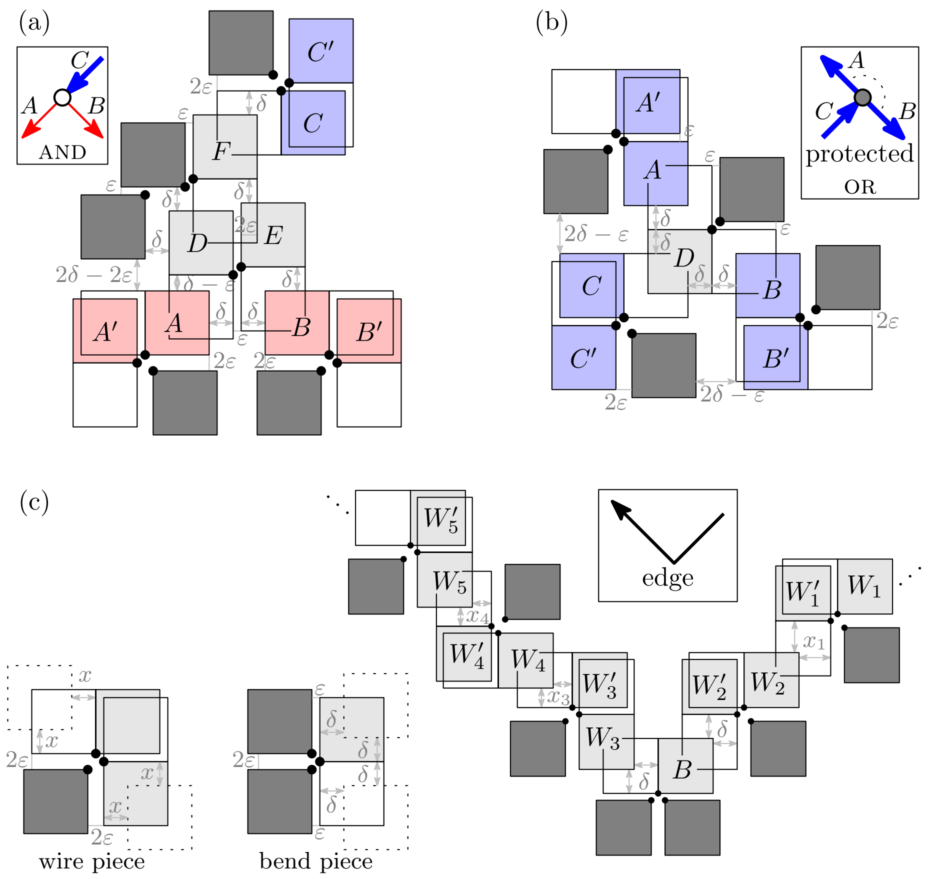


Fig. 3. Our gadgets simulating the (a) AND vertex, (b) protected OR vertex, and (c) edge shown in the insets. For the edge gadget, the left side shows its building pieces. They may be rotated arbitrarily in 90° increments, and "connected" together along the dotted lines. The right side shows an example that can be built. The gadgets work for square labels of side length 1 (depicted) up to $1 + \delta - \varepsilon$. In the figure, $\delta = 3/8$ and $\varepsilon = 1/8$. The distance marked x may be varied along an edge gadget in order to make it line up correctly with vertex gadgets. For the 4-position model, $x \in [\delta - \varepsilon, 1)$; for the 2-slider and 4-slider models, $x \in [\delta - \varepsilon, 1 - 3\varepsilon)$.

gray labels, which restrict the amount of usable space for the red and blue labels inside the vertex gadget.

Figure 3 shows our vertex and edge gadgets in more detail. As mentioned, the blockers restrict the colored labels marked A, B, and C to two possible positions each in the 4-position model. We call the label position closest to the center of the vertex gadget *inward*, and the other *outward*. In the figure, labels A and B are placed inward, and label C is placed outward. In the slider models, labels can take on any position in between these two extremes, but we may assume that only a very small range of positions is actually used. Consider, for example, label A in Fig. 3(b). Without moving label A', we can only move A up by at most ε, or left by at most 2ε. We refer to positions for A from this range as *inward*, and define the term similarly for B and C. If (and only if) A' is moved left by at least $1 - \varepsilon$, then A can move further upward. We may then move A all the way to its uppermost position, and there is no reason not to do so. Thus we may define *outward* as in the 4-position model.

Our edge gadgets ensure that their two incident vertex gadgets cannot both have their corresponding label placed outward simultaneously. Both *can* be placed inward simultaneously, however, which corresponds to the NCL edge having an indeterminate orientation. Modifying the definition of a legal configuration to allow such edges does not meaningfully change NCL, though, as any inflow constraints that were already satisfied would remain so after arbitrarily orienting such edges. With these definitions there is a direct correspondence between static and dynamic labelings for our construction on the one hand, and legal configurations and moves on a constraint graph on the other hand. This is expressed in the following theorem, the proof of which we omit due to space constraints.

Theorem 1. *Let G be a planar constraint graph with n vertices, and let ε and δ be two real numbers with $0 < \varepsilon \leqslant \delta < 1 - 3\varepsilon$. One can then construct a point set $P = P(G, \delta, \varepsilon)$ in polynomial time that has the following properties for any $s \in [1, 1 + \delta - \varepsilon]$:*
- *the size of P, and the coordinates of its points, are polynomially bounded in n,*
- *for any legal configuration of G there is an overlap-free static labeling of P with $s \times s$ square labels in the 4-position model and vice versa,*
- *there is a sequence of moves transforming one legal configuration of G into another if and only if there is a dynamic labeling transforming the corresponding static labelings of P into each other.*

When $\delta < 1/3 - 4\varepsilon/3$, the same results hold for the 2- and 4-slider models.

3 Hardness of Dynamic Point Labeling

In this section we will define a number of dynamic point-labeling problems. The input to all of them is a dynamic point set P, which specifies for each point $p \in P$ an arrival and departure time, as well as a continuous trajectory. In some problems these changes to the point set may be fairly arbitrary, in others they must be the result of the user panning, rotating, or zooming their viewport of a static point set. In all cases we seek a dynamic labeling $\mathcal{L}$ of P for a given time interval $[a, b]$. That is, $\mathcal{L}(t)$ must be a static labeling of $P(t)$ for all $t \in [a, b]$, and each of $\mathcal{L}$'s labels must move continuously over time.

Various optimization questions may be formulated for dynamic labeling. We may disallow label overlap entirely, and then seek a dynamic labeling that labels as many points as possible for as long as possible. Alternatively, perhaps we wish to label all points at all times, and wish to have as little label overlap as possible. Whatever the case may be, labeling all points at all times without any overlap is likely the most desirable outcome. Unfortunately, we will see that deciding whether such a solution exists is already strongly PSPACE-complete, even if we drastically restrict the dynamic nature of the point set. Thus, all optimization problems of this kind are strongly PSPACE-hard.

Theorem 2. *The following decision problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A dynamic point set P (with given trajectories, arrival times, and departure times), and numbers a and b with $a < b$.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

Additionally, unless P = PSPACE, the maximum label size for which there is such an $\mathcal{L}$ cannot be $(4/3 - \varepsilon')$-approximated in polynomial time for any $\varepsilon' > 0$. For the 4-position model, it cannot even be $(2-\varepsilon')$-approximated. All of the above remains true when

– *all points are stationary, and during $[a, b]$ two points are removed and two points are added,*
– *no points are added or removed, and all points move at the same, constant speed along parallel (but possibly opposite), straight-line trajectories, or*
– *no points are added or removed, and all but two points are stationary.*

Proof. We omit the proof of membership in PSPACE and only show strong PSPACE-hardness. To do so we reduce from edge-to-edge NCL. Thus we are given a constraint graph G with orientations for two edges e_A and e_B, and want to decide whether there is a sequence of moves on G starting from a legal configuration $\mathcal{C}_A$ where e_A has its specified orientation and ending in a legal configuration $\mathcal{C}_B$ where e_B has its specified orientation. We start with the point set $P = P(G, \delta, \varepsilon)$ of Theorem 1 (for δ and ε to be determined below), and slightly modify the edge gadgets for e_A and e_B as follows. Pick one blocker $q \in P$ in the edge gadget for e_B, and suppose $p \in P$ is the point for which q blocks some label candidates. Now make q move at a constant speed of $v = 3\varepsilon/(b - a)$ towards the nearest non-blocked candidate of p, as in Fig. 4(a)–(b). In the edge gadget for e_A we select a blocker q' and non-blocker p' in the same way. Now, however, we alter

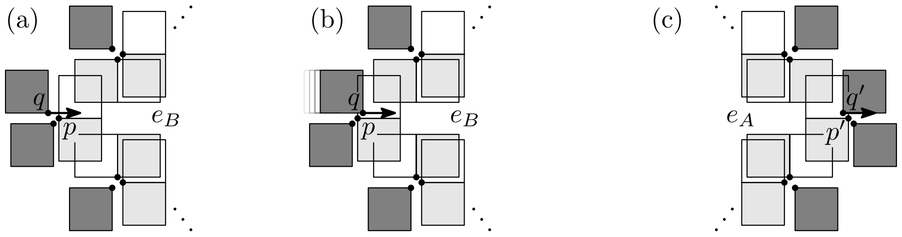


Fig. 4. Our reduction from edge-to-edge NCL to dynamic labeling of moving points. (a) The modified gadget for edge e_B at time a. (b) The modified gadget for edge e_B at time b. (c) The modified gadget for edge e_A at time a.

the position of q' at time a to be inside the non-blocked label candidate of p', and make q' move at speed v in the direction of its original position, as in Fig. 4(c). This causes e_A to be constrained to a single orientation during the time interval $[a, a + \Delta)$, and e_B during the time interval $(b - \Delta, b]$, where $\Delta = (b - a)/3$. We may achieve the same result with all points moving on parallel trajectories by being careful how we lay out the edge gadgets on the grid. We can then ensure that the two blockers move in the same direction (as they do in the figure), and may reduce their speeds to $v/2$, while moving all other points at speed $v/2$ in the opposite direction. Alternatively, we may keep all points stationary and at time $a + \Delta$ remove q' from P and re-insert it at its new location, and do the same for q at time $b - \Delta$. In all cases, the desired dynamic labeling exists if and only if there is a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$, and deciding the latter is PSPACE-hard. All coordinates are polynomially bounded in the size of G, making the decision problem strongly PSPACE-hard. The construction works for identical square labels with a side length in the range $[1, 1 + \delta - \varepsilon]$. We must have $0 < \varepsilon \leqslant \delta < 1 - 3\varepsilon$ for the 4-position model, so pick $\varepsilon = \varepsilon'/4$ and $\delta = 1 - \varepsilon'$ to obtain the hardness-of-approximation result. For the 2-slider and 4-slider models $\delta < 1/3 - 4\varepsilon/3$ must hold, so pick $\delta = 1/3 - \varepsilon'$ instead. □

In some applications it may be that the static labelings $\mathcal{L}(a)$ and/or $\mathcal{L}(b)$ are already given. The trajectories of the points may only become known through periodic updates, for example. If we know the trajectories up to time b at time a, then we have to find a dynamic labeling that extends from the currently displayed labeling $\mathcal{L}(a)$. As another example, $P(a)$ and $P(b)$ may represent two stationary configurations of the points for which we have computed high-quality labelings $\mathcal{L}(a)$ and $\mathcal{L}(b)$. As the points are smoothly transitioned from one configuration to the other, we then want to do likewise with the labeling. These variants of the problem are even harder, in the sense that less dynamism is needed in P to prove them PSPACE-hard.

Theorem 3. *(i) When $\mathcal{L}(a)$ or $\mathcal{L}(b)$ is given, Theorem 2 holds even when*
- *all points are stationary, one point gets removed and one point gets added,*
- *no points are added or removed, and all points move at the same, constant speed along parallel (but possibly opposite), straight-line trajectories, or*
- *no points are added or removed, and all points but one are stationary.*

(ii) When both $\mathcal{L}(a)$ and $\mathcal{L}(b)$ are given the theorem holds even for static point sets (all points stationary, none added or removed).

Proof (sketch). (i) Follow the proof of Theorem 2 but reduce from configuration-to-edge NCL and modify only one edge gadget instead of two. (ii) Follows directly from Theorem 1. □

In interactive maps, users are presented with a rectangular *viewport* V showing a portion of a larger map. By panning, rotating, and/or zooming the map, the user controls which portion of the map is displayed at any given time. The task of labeling the points inside V can be seen as a special case of labeling dynamic point sets. Continuous panning, rotation, and zooming of the map may cause points to enter or leave V at its boundary, and causes all points within V to move on continuous trajectories. We will require only the points inside V to be labeled, and these labels must be fully contained in V. Points outside of V need not be labeled, but we may wish to do so in order to ensure a smooth dynamic labeling. Otherwise a label would have to instantly appear or disappear whenever its point hits the boundary of V. Regardless of whether we allow such "popping" of labels, the question of whether a dynamic labeling can label all points in the viewport without overlap is strongly PSPACE-complete.

Theorem 4. *The following decision problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider label models.*

Given: *A closed rectangle V, a set of points P being panned and/or rotated, and two numbers a and b with $a < b$.*
Decide: *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels $P(t) \cap V$ with non-overlapping unit-square labels inside V for all $t \in [a, b]$.*

This is true regardless of whether the labelings $\mathcal{L}(a)$ and/or $\mathcal{L}(b)$ are given, and whether points on the boundary of V may instantly lose/gain their labels, even
- *when the points are only panning, along a straight line at constant speed, or*
- *when the points are only rotating, in a fixed direction at constant speed.*

Proof. We prove PSPACE-hardness for the 4-slider model, with neither $\mathcal{L}(a)$ nor $\mathcal{L}(b)$ being given. The remaining results can then be derived by similar techniques as in previous theorems. Our reduction is from edge-to-edge NCL, where we are given a constraint graph G and orientations for two of its edges e_A and e_B. We construct the point set $P = P(G, \delta, \varepsilon)$ as usual, but this time we lay it out on the grid in such a way that e_A is on the far left and e_B is on the far right, as in Fig. 5. While this is always possible, it may introduce edge crossings. However, Hearn and Demaine show [6] how to construct a planar "cross-over" in NCL that functions as two crossing edges. This cross-over uses only AND and protected OR vertices, so we may emulate it using our gadgets.

Next, we add two additional points q and q', with q to the left of e_A by $1-2\varepsilon$ and q' to the right of e_B by $1 - 3\varepsilon$, as in Fig. 5. We construct a viewport

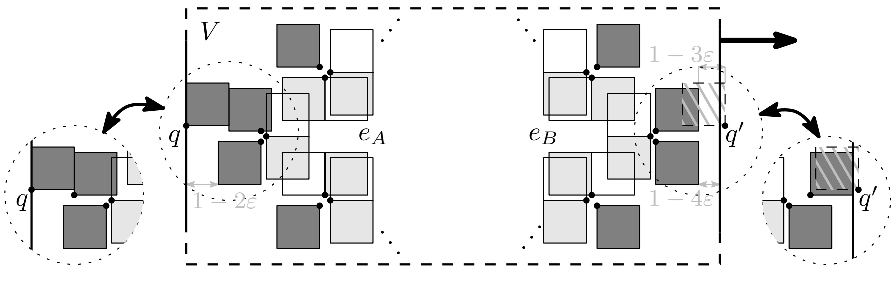


Fig. 5. Our reduction from edge-to-edge NCL to dynamic labeling under panning. The main figure shows the construction for the 4-slider model, the circular insets at the left and right show how to modify the construction for the 4-position model.

rectangle V with its left side containing q, and its right side ε to the left of q'. This forces us to label q in a way that constrains e_A to a single orientation. Now suppose the points move to the left (that is, the view pans to the right). This lifts this constraint on e_A, as we may then move q's label out of V (either immediately, or over a short time interval). Panning may continue for a distance of $1 - \varepsilon$ before disturbing the functioning of the edge gadget for e_A. However, already after moving by ε are we required to start labeling point q' in a way that constrains e_B to a single orientation. Thus there exists a dynamic labeling of P for this panning motion if and only if there exists a sequence of moves in G starting with e_A in its specified orientation and ending with e_B in its specified orientation. This makes the problem PSPACE-hard for the 4-slider model. For a horizontal 2-slider model we may use the exact same construction; for a vertical 2-slider model we simply rotate the construction by 90°, placing e_A and e_B at the top and bottom. For the 4-position model, the construction needs a small modification so that labels always contain their point on a corner. This modification is depicted in the circular insets of Fig. 5.

Now suppose that instead of panning, P is rotating. We modify V so that q and q' both lie below its center c. Rotating P clockwise around c then moves q out of V, and q' towards V, similar as for panning. However, this now also changes the horizontal and vertical distances between the points of the gadgets. As long as these changes are less than ε, all gadgets will still work. Thus, a sufficiently small rotation will not disturb our reduction. To ensure q' ends up inside V during this rotation, decrease the initial distance between V and q' appropriately by moving the right side of V slightly farther to the right. □

In contrast to panning and rotation, PSPACE-hardness for zooming depends on whether a static labeling is given and which one. If $\mathcal{L}(t)$ is given, and during $[a, b]$ the view is maximally zoomed out at time $t \in [a, b]$, then the problem is trivial. When zooming in further all inter-point distances increase, and no new points enter V, so the given labeling can be used throughout $[a, b]$. If we are not given any labelings, the problem is "merely" NP-complete: determine the time $t \in [a, b]$ where the view is maximally zoomed out, and solve the static labeling problem for $P(t)$. The remaining cases, however, are PSPACE-complete.

Theorem 5. *Theorem 4 also applies to zooming, assuming at least one static labeling is given, and not the one at which the view is maximally zoomed out during $[a, b]$. This is true even when only zooming in or only zooming out, at constant speed.*

Proof. Consider again the construction in Fig. 5. Suppose we delete the point q', and then start zooming in. This will cause q to move out of V, and the constraint on e_A will be lifted. Conversely, suppose we delete point q, and then start zooming out. This will cause q' to move into V, adding a constraint on e_B. Thus we can reduce configuration-to-edge NCL to dynamic labeling under zooming. As with rotation, we will have to be careful that the changes to inter-point distances do not disturb the gadgets. The solution is the same: move the right side of V closer to q' so that we do not have to zoom so far as to disturb the gadgets. □

4 Conclusion

We have examined the following dynamic point labeling problem. Given a set of points moving along continuous trajectories, and where points may be added and removed over time, is there a smooth function from time to static point labelings? For the 4-position, 2-slider, and 4-slider models this problem is strongly PSPACE-complete. In addition, finding the maximum label size at which such a labeling does exist admits no PTAS, unless P = PSPACE. For the 4-position model a 2-approximation is the best that can be hoped for, and for the other models a 4/3-approximation. The PSPACE-completeness results also apply for the special case where the points are panned, rotated, or zoomed inside a fixed viewport. For this case our constructions have less "wiggle room", meaning our hardness-of-approximation results do not apply. The wiggle room is still non-zero, however, meaning that a PTAS would still imply P = PSPACE.

It remains to examine other label models such as the 1- and 2-position models and the 1-slider model. For static labeling their corresponding decision problems are easily solved in polynomial time. On the other hand, the number-maximization problem is still NP-hard for them. Perhaps the complexity landscape of dynamic labeling is similar. Most importantly, perhaps, is the continued pursuit of approximation algorithms for optimization problems in dynamic point labeling. While there are heuristics for both number maximization [8] and free-label maximization [2], no guaranteed approximation ratios have been achieved.

References

1. Been, K., Nöllenburg, M., Poon, S.-H., Wolff, A.: Optimizing active ranges for consistent dynamic map labeling. Comput. Geom. Theory Appl. 43(3), 312–328 (2010)
2. de Berg, M., Gerrits, D.H.P.: Labeling moving points with a trade-off between label speed and label overlap. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 373–384. Springer, Heidelberg (2013)
3. Biedl, T., Kant, G.: A better heuristic for orthogonal graph drawings. Comput. Geom. Theory Appl. 9(3), 159–180 (1998)
4. Formann, M., Wagner, F.: A packing problem with applications to lettering of maps. In: Proc. 7th ACM Sympos. Comput. Geom. (SoCG 1991), pp. 281–288 (1991)
5. Gemsa, A., Nöllenburg, M., Rutter, I.: Consistent labeling of rotating maps. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 451–462. Springer, Heidelberg (2011)
6. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoret. Comput. Sci. 343(1-2), 72–96 (2005)
7. van Kreveld, M., Strijk, T., Wolff, A.: Point labeling with sliding labels. Comput. Geom. Theory Appl. 13, 21–47 (1999)
8. Vaaraniemi, M., Treib, M., Westermann, R.: Temporally coherent real-time labeling of dynamic scenes. In: Proc. 3rd Internat. Conf. Computing for Geospatial Research and Applications, COM.Geo 2012, article no. 17 (2012)
9. Wolff, A., Strijk, T.: The Map Labeling Bibliography (2009), `http://liinwww.ira.uka.de/bibliography/Theory/map.labeling.html`

Unsatisfiable CNF Formulas contain Many Conflicts

Dominik Scheder[⋆,⋆⋆,⋆⋆⋆]

Aarhus University

Abstract. A pair of clauses in a CNF formula constitutes a conflict if there is a variable that occurs positively in one clause and negatively in the other. A CNF formula without any conflicts is satisfiable. The Lovász Local Lemma implies that a CNF formula with clauses of size exactly k (a *k-CNF formula*), is satisfiable unless some clause conflicts with at least $\frac{2^k}{e}$ clauses. It does not, however, give any good bound on how many conflicts an unsatisfiable formula has globally. We show here that every unsatisfiable k-CNF formula requires $\Omega(2.69^k)$ conflicts and there exist unsatisfiable k-CNF formulas with $O(3.51^k)$ conflicts.

1 Introduction

A boolean formula in conjunctive normal form (short a *CNF formula*) is a conjunction (AND) of *clauses*, which are disjunctions of literals. A *literal* is either a boolean variable x or its negation $\bar{x}$. SAT, the problem of deciding whether a CNF formula is satisfiable is a central problem in theoretical computer science, and was one of the first problems to be proven NP-complete. How can a CNF formula become unsatisfiable? Roughly speaking, there are two possibilities: Either some clause itself is impossible to satisfy – this is only the case for the empty clause. Or, each clause is individually satisfiable, but there are conflicts between the clauses, making it impossible to satisfy all of them simultaneously. When we consider k-CNF formulas, where each clause consists of exactly k literals (we require that literals in a clause do not repeat), then each clause is extremely easy to satisfy: Of the 2^k possible truth assignments to its variables, all but one satisfy it. If a k-CNF formula is unsatisfiable, we expect it to have many conflicts.

To give a formal setup, we say two clauses *conflict* if there is at least one variable that appears positively in one clause and negatively in the other. For example, the two clauses $(x \vee y)$ and $(\bar{x} \vee u)$ conflict. Similarly, $(x \vee y)$ and

[⋆] The author acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which this work was performed.

[⋆⋆] Research was supported by the SNF Grant 200021-118001/1.

[⋆⋆⋆] The author acknowledges support from the Simons Institute for the Theory of Computing, UC Berkeley.

$(\bar{x} \vee \bar{y})$ do. Suppose F is a CNF formula without the empty clause, and without any conflicts. Then clearly F is satisfiable. For a formula F we define the *conflict graph* $CG(F)$, whose vertices are the clauses of F, and two clauses are connected by an edge if they conflict. $\Delta(F)$ denotes the maximum degree of $CG(F)$ and $e(F)$ the number of conflicts in F, i.e. the number of edges in $CG(F)$. Our above observation now reads as follows: If F does not contain the empty clause, and $e(F) = 0$, then F is satisfiable. In fact, any k-CNF formula is satisfiable unless $\Delta(F)$ and $e(F)$ are large. How large? A quantitative result follows from the Lopsided Lovász Local Lemma [1,2,3]: A k-CNF formula F is satisfiable unless some clause conflicts with $\frac{2^k}{e}$ or more clauses, i.e., unless $\Delta(F) \geq \frac{2^k}{e}$. Up to a constant factor, this is tight: Consider the formula containing all 2^k clauses over the variables $x_1, \ldots, x_k$. We call this a *complete k-CNF formula* and denote it by $\mathcal{K}_k$. It is unsatisfiable, and $\Delta(\mathcal{K}_k) = 2^k - 1$.

As its name suggests, the Lopsided Lovász Local Lemma implies a *local* result: A k-CNF formula F is satisfiable, unless *somewhere* in F there are many conflicts. We want to obtain a *global* result: F is satisfiable unless the total number of conflicts is *very* large. We define two functions:

$$lc(k) := \max\{d \in \mathbb{N}_0 \mid \text{ every } k\text{-CNF formula } F \text{ with } \Delta(F) \leq d \text{ is satisfiable}\}\ ,$$
$$gc(k) := \max\{d \in \mathbb{N}_0 \mid \text{ every } k\text{-CNF formula } F \text{ with } e(F) \leq d \text{ is satisfiable}\}\ .$$

The abbreviations lc and gc stand for *local conflicts* and *global conflicts*, respectively. From the above discussion, $\frac{2^k}{e} - 1 \leq lc(k) \leq 2^k - 2$, hence we know $lc(k)$ up to a constant factor. In contrast, it does not seem to be easy to prove nontrivial upper and lower bounds on $gc(k)$. Let us see what we get: Surely, $gc(k) \geq lc(k) \geq \frac{2^k}{e} - 1$. For an upper bound, $gc(k) \leq e(\mathcal{K}_k) - 1 = \binom{2^k}{2} - 1$. Ignoring constant factors, $gc(k)$ lies somewhere between 2^k and 4^k. This leaves much space for improvement. In [4], Zumstein and I proved that $gc(k) \in \Omega(2.27^k)$ and $gc(k) \leq \frac{4^k}{\log^3 k} k$. In this paper, we significantly improve upon these bounds. Somehow surprisingly, $gc(k)$ is exponentially smaller than 4^k.

Theorem 1. *Any unsatisfiable k-CNF formula contains $\Omega\left(2.69^k\right)$ conflicts. On the other hand, there is an unsatisfiable k-CNF formula with $O\left(3.51^k\right)$ conflicts.*

We obtain the lower bound by a more sophisticated application of the idea used in [4]. The upper bound follows from a construction that is partially probabilistic, and inspired in parts by Erdős' construction in [5] of small k-uniform hypergraphs that are not 2-colorable.

1.1 Related Work

Let F be a CNF formula and u be a literal. We write $\mathrm{occ}_F(u) := |\{C \in F \mid u \in C\}|$. For a variable x, we write $d_F(x) = \mathrm{occ}_F(x) + \mathrm{occ}_F(\bar{x})$. So $d_F(x)$, the *degree* of x, counts the number of clauses containing the variable x, irrespective of its polarity. We write $d(F) = \max_x d_F(x)$. It is easy to see that for a k-CNF formula,

$\Delta(F) \leq k(d(F)-1)$. We define

$$f(k) := \max\{d \in \mathbb{N}_0 \mid \text{ every } k\text{-CNF formula } F \text{ with } d(F) \leq d \text{ is satisfiable}\} .$$

The function $f(k)$ has been subject of some research. By an application of Hall's Theorem, Tovey [6] showed that every k-CNF formula F with $d(F) \leq k$ is satisfiable, hence $f(k) \geq k$. Later, Kratochvíl, Savický and Tuza [7] showed that $f(k) \geq \frac{2^k}{ek}$: In our terminology, they showed that $lc(k) \geq \frac{2^k}{e} - 1$ and then used the fact that $\Delta(F) \leq k(d(F)-1)$. As for an upper bound, in [7] the authors show that $f(k) \leq 2^{k-1} - 2^{k-4} - 1$. This was improved by Savický and Sgall [8] to $f(k) \in O(k^{-0.26}2^k)$, by Hoory and Szeider [9] to $f(k) \in O\left(\frac{\log(k)2^k}{k}\right)$, and only recently, by Gebauer [10] to $f(k) \leq \frac{2^{k+2}}{k} - 1$ clauses, closing the gap between lower and upper bound on $f(k)$ up to a constant factor. Finally, Gebauer, Szabó, Tardos [11] proved that $f(k) = (1 \pm o(1))2^{k+1}/ek$, which even determines the constant factor.

1.2 Conflicts Generated by a Single Variable

Let F be a CNF formula and x a variable. Every clause containing x conflicts with every clause containing $\bar{x}$, thus $e(F) \geq \mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x})$. In fact,

$$e(F) \geq \frac{1}{k} \sum_x \mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x}) \tag{1}$$

where the $\frac{1}{k}$ comes from the fact that each conflict might be counted up to k times, if two clauses contain several complementary literals. By [7], every unsatisfiable k-CNF formula F contains a variable x with $d_F(x) \geq \frac{2^k}{ek}$. If this variable is *balanced*, i.e. $\mathrm{occ}_F(x)$ and $\mathrm{occ}_F(\bar{x})$ are both at least $\frac{2^k}{\mathrm{poly}(k)}$, then $e(F) \geq \frac{4^k}{\mathrm{poly}(k)}$. Indeed, in the formulas constructed in [10], all variables are balanced. The same holds for the complete k-CNF formula $\mathcal{K}_k$. Thus, it might be the case that in every unsatisfiable k-CNF formula, there is a single variable that already generates many conflicts:

Conjecture 1. There exists a number $a > 2$ such that every unsatisfiable k-CNF formula F contains a variable x such that $\mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x}) \geq \Omega\left(a^k\right)$.

We do not know whether this conjecture is true. However, we will give non-trivial *upper* bounds on $\mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x})$:

Theorem 2. *For all sufficiently large k, there is an unsatisfiable k-CNF formula with $\mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x}) \leq 3.01^k$ for all variables x.*

2 Notation and Tools

Throughout the paper, we regard formulas as sets of clauses and clauses as sets of literals. This is purely to simplify notation. For a truth assignment α and a clause C, we will write $\alpha \models C$ if α satisfies C. Similarly $\alpha \not\models C$ if it does not. If α satisfies a formula F, we write $\alpha \models F$.

We will state a version of the Lopsided Lovász Local Lemma formulated in terms of satisfiability. For a derivation of this version see [12].

Lemma 1 (SAT Version of the Lopsided Lovász Local Lemma). *Let F be a CNF formula not containing the empty clause. Sample a truth assignment α by independently setting each variable x to* `true` *with $p(x) \in [0,1]$. If for any clause $C \in F$, it holds that*

$$\sum_{D \in F\colon\, C \text{ and } D \text{ conflict}} \Pr[\alpha \not\models D] \leq \frac{1}{4} \tag{2}$$

then F is satisfiable.

In our proofs, it will be difficult to apply Lemma 1 to a formula F which we want to prove satisfiable. Instead, we apply it to a formula F' we obtain from F in the following way:

Definition 1. *Let F be a CNF formula. A* truncation *of F is a CNF formula F' that is obtained from F by deleting some literals from some clauses.*

For example, $(x \vee y) \wedge (\bar{y} \vee z)$ is a truncation of $(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z)$. A truncation of a k-CNF formula is not a k-CNF formula anymore. It is easy to see that any truth assignment satisfying a truncation F' of F also satisfies F. In our proofs, we will often find it easier to apply Lemma 1 to a special truncation of F than to F itself. We need a technical lemma on the binomial coefficient.

Lemma 2. *Let $a, b \in \mathbb{N}$ with $b/a \leq 0.75$. Then*

$$\frac{a^b}{b!} \geq \binom{a}{b} > \frac{a^b}{b!} e^{-b^2/a} .$$

Proof. The upper bound is trivial and true for all a, b. The lower bound follows like this.

$$\binom{a}{b} = \frac{a(a-1)\cdots(a-b+1)}{b!} = \frac{a^b}{b!} \prod_{j=0}^{b-1} \frac{a-j}{a} > \frac{a^b}{b!} e^{-\frac{2}{a}\sum_{j=0}^{b-1} j} > \frac{a^b}{b!} e^{-b^2/a} ,$$

where we used the fact that $1 - x > e^{-2x}$ for $0 \leq x \leq 0.75$. □

3 Upper Bounds – Probabilistic Constructions of Unsatisfiable Formulas

As we have argued in Section 1.2, in order to improve significantly upon the upper bound $gc(k) \leq 4^k$, we must construct a formula that is very unbalanced, i.e. $\mathrm{occ}_F(x)$ is exponentially larger than $\mathrm{occ}_F(\bar{x})$. The central idea is that we do not construct an unsatisfiable k-CNF formula, but allow certain clauses to be smaller. In a second step, we expand these clauses to size k.

Definition 2. *Let F be a CNF formula with clauses of size at most k. For each k'-clause C with $k' < k$, construct a complete $(k - k')$-CNF formula $\mathcal{K}_{k-k'}$ over $k - k'$ new variables $y_1^C, \ldots, y_{k-k'}^C$. We replace C by $C \vee \mathcal{K}_{k-k'}$. Using distributivity, we expand it into a k-CNF formula G called the* k-CNFification *of F.*

For example, the 3-CNFification of $(x \vee y) \wedge (\bar{x} \vee y \vee z)$ is $(x \vee y \vee y_1) \wedge (x \vee y \vee \bar{y}_1) \wedge (\bar{x} \vee y \vee z)$. It is easy to see that a truth assignment satisfies F if and only if it satisfies its k-CNFification G.

Definition 3. *Let $\ell, k \in \mathbb{N}_0$. An (ℓ, k)-CNF formula is a formula consisting of ℓ-clauses containing only positive literals, and k-clauses containing only negative literals.*

If F is an (ℓ, k)-CNF formula, we write $F = F^+ \wedge F^-$, where F^+ consists of purely positive ℓ-clauses and F^- of purely negative k-clauses.

Proposition 1. *Let $\ell \leq k$, and let $F = F^+ \wedge F^-$ be an (ℓ, k)-CNF formula. Let G be the k-CNFification of F. Then*

(i) $e(G) \leq 4^{k-\ell}|F^+| + 2^{k-\ell}|F^+| \cdot |F^-|$,
(ii) $\mathrm{occ}_G(x) \cdot \mathrm{occ}_G(\bar{x}) \leq \max\{4^{k-\ell}, 2^{k-\ell}|F^+| \cdot |F^-|\}$ *for every variable x.*

Proof. Every edge in $CG(F)$ runs between a positive ℓ-clause C and a negative k-clause D. Thus, $e(F) \leq |F^+| \cdot |F^-|$. In G, this edge is replaced by $2^{k-\ell}$ edges, since C is replaced by $2^{k-\ell}$ copies. Replacing C by $2^{k-\ell}$ copies introduces less than $4^{k-\ell}$ edges. This proves (i). To prove (ii), there are two cases. First, if x appears in F, then $\mathrm{occ}_G(\bar{x}) = \mathrm{occ}_F(\bar{x})$ and $\mathrm{occ}_G(x) = \mathrm{occ}_F(x) 2^{k-\ell}$, thus $\mathrm{occ}_G(x)\mathrm{occ}_G(\bar{x}) \leq 2^{k-\ell}|F^+| \cdot |F^-|$. Second, if x does not appear in F, it has been introduced in the k-CNFification. Then $\mathrm{occ}_G(x) = \mathrm{occ}_G(\bar{x}) = 2^{k-\ell-1}$, and $\mathrm{occ}_G(x) \cdot \mathrm{occ}_G(\bar{x}) \leq 4^{k-\ell}$. □

We will explore for which values of $|F^+|$ and $|F^-|$ there are unsatisfiable (ℓ, k)-CNF formulas. Then we use Proposition 1 to derive the upper bounds of Theorem 1 and Theorem 2.

Lemma 3. *(i) For any $\rho \in (0, 1)$, there is a constant c such that for all k and $\ell \leq k$, there exists an unsatisfiable (ℓ, k)-CNF formula $F = F^+ \wedge F^-$ with $|F^-| \leq ck^2\rho^{-k}$ and $|F^+| \leq ck^2(1 - \rho)^{-\ell}$.*

(ii) Let $F = F^+ \wedge F^-$ be an (ℓ, k)-CNF formula. If there is a $\rho \in (0, 1)$ such that $|F^+| < \frac{1}{2}(1 - \rho)^{-\ell}$ and $|F^-| < \frac{1}{2}\rho^{-k}$, then F is satisfiable.

Proof. We begin with (ii), which is easier. Sample a truth assignment α by setting each variable independently to `true` with probability ρ. For a negative k-clause C, it holds that $\Pr[\alpha \not\models C] = \rho^k$. Similarly, for a positive ℓ-clause D, $\Pr[\alpha \not\models D] = (1 - \rho)^\ell$. Hence the expected number of clauses in F that are unsatisfied by α is $\rho^k|F^-| + (1-\rho)^\ell|F^+| < \frac{1}{2} + \frac{1}{2} = 1$. Therefore, with positive probability α satisfies F.

For (i), we choose a set $V = \{x_1, \ldots, x_n\}$ of $n = k^2$ variables. Let c be a constant, to be determined later. We form F^- by sampling, with replacement, $ck^2\rho^{-k}$ negative k-clauses from all $\binom{n}{k}$ possible. Similarly, we form F^+ by sampling $ck^2(1-\rho)^{-\ell}$ positive ℓ-clauses. We claim that for a suitable choice of c this formula is unsatisfiable with high probability. Let α be any truth assignment. There are two cases. First, suppose α sets at least ρn variables to `true`. For a random negative clause C,

$$\Pr[\alpha \not\models C] \geq \frac{\binom{\rho n}{k}}{\binom{n}{k}} \geq \frac{\frac{(\rho n)^k}{k!} \cdot e^{-k^2/(\rho n)}}{\frac{n^k}{k!}} = \rho^k e^{-1/\rho} = c'\rho^k$$

By independence, $\Pr[\alpha \models F^-] \leq (1 - c'\rho^k)^{ck^2\rho^{-k}} < e^{-cc'k^2}$. Second, suppose α sets at most ρn variables to `true`. By a similar argument, $\Pr[\alpha \models F^+] \leq (1 - c''(1-\rho)^\ell)^{ck^2(1-\rho)^{-\ell}} < e^{-cc''k^2}$. For suitable c, we obtain $\Pr[\alpha \models F] < e^{-k^2} = e^{-n}$ for any α. The expected number of satisfying assignments of F is thus less than $2^n e^{-n} < 1$. With high probability F is unsatisfiable. □

It should be pointed out that for $k = \ell$, an (ℓ, k)-CNF formula is just a monotone k-CNF formula. The size of a smallest unsatisfiable monotone k-CNF formula is the same – up to a factor of at most 2 – as the minimum number of hyperedges in a k-uniform hypergraph that is not 2-colorable. In 1963, Erdős [13] raised the question what this number is, and proved a 2^{k-1} lower bound (this is easy, simply choose a random 2-coloring). One year later, he [5] gave a probabilistic construction of a non-2-colorable k-uniform hypergraph using $ck^2 2^k$ hyperedges. For $\ell = k$ and $\rho = \frac{1}{2}$, the above proof is basically the same as Erdős' proof.

Proof (Proof of Theorem 2). Combining Lemma 3 and Proposition 1, we conclude that for any $\rho \in (0,1)$ and $0 \leq \ell \leq k$, there is an unsatisfiable k-CNF formula F with

$$\mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x}) \leq \max\{4^{k-\ell}, 2^{k-\ell}c^2k^4\rho^{-k}(1-\rho)^{-\ell}\} ,$$

for every variable x. The constant c depends on ρ, but not on k or ℓ. The term $\rho^{-k}(1-\rho)^{-\ell}$ is minimized for $\rho = \frac{k}{k+\ell}$. Choosing $\ell = \lceil 0.2055k \rceil$, we get $\rho \approx 0.83$ and $\mathrm{occ}_F(x) \cdot \mathrm{occ}_F(\bar{x}) \in O(3.01^k)$. □

Proof (Proof of the upper bound of Theorem 1). As in the previous proof, Proposition 1 together with Lemma 3 yield an unsatisfiable k-CNF formula F with

$$e(F) \leq 4^{k-\ell}ck^2(1-\rho)^{-\ell} + 2^{k-\ell}c^2k^4\rho^{-k}(1-\rho)^{-\ell} .$$

For $\rho \approx 0.6298$ and $\ell = \lceil 0.333k \rceil$, we obtain $e(F) \in O(3.51^k)$. □

4 A Lower Bound on the Number of Global Conflicts

Proof (of the lower bound in Theorem 1). Let F be an unsatisfiable k-CNF formula and let $e(F)$ be the number of conflicts in F. We will show that $e(F) \in \Omega\left(2.69^k\right)$. In the proof, x denotes a variable and u a positive or negative literal. We assume $\mathrm{occ}_F(\bar{x}) \le \mathrm{occ}_F(x)$ for all variables x. We can do so since otherwise we just replace x by $\bar{x}$ and vice versa. This changes neither $e(F)$, nor satisfiability of F. Also we can assume that $\mathrm{occ}_F(x)$ and $\mathrm{occ}_F(\bar{x})$ are both at least 1, if x occurs in F at all. For x, we define

$$p(x) := \max\left\{\frac{1}{2}, \sqrt[k]{\frac{\mathrm{occ}_F(x)}{16e(F)}}\right\} .$$

We define a random truth assignment α by setting x to `true` with probability $p(x)$, independently for each variable. Since $\mathrm{occ}_F(u) \le e(F)$, we have $p(x) \le 1$. We set $p(\bar{x}) = 1 - p(x)$. By definition $p(x) \ge p(\bar{x})$. Let us list some properties of this distribution. First, if $p(u) < \frac{1}{2}$ for some literal u, then u is a negative literal $\bar{x}$, and $p(x) = \sqrt[k]{\frac{\mathrm{occ}_F(x)}{16e(F)}} > \frac{1}{2}$. Second, if $p(u) = \frac{1}{2}$, then both $\sqrt[k]{\frac{\mathrm{occ}_F(x)}{16e(F)}} \le \frac{1}{2}$ and $\sqrt[k]{\frac{\mathrm{occ}_F(\bar{x})}{16e(F)}} \le \frac{1}{2}$ hold. We distinguish two types of clauses: *Bad* clauses, which contain at least one literal u with $p(u) < \frac{1}{2}$, and *good* clauses, which contain only literals u with $p(u) \ge \frac{1}{2}$.

Lemma 4. *Let $\mathcal{B} \subseteq F$ denote the set of bad clauses. Then $\sum_{C\in\mathcal{B}} \Pr\,[\alpha \not\models C] \le \frac{1}{8}$.*

Proof. For each clause $C \in \mathcal{B}$, let u_C be the literal in C minimizing $p(u)$, breaking ties arbitrarily. This means $\Pr[\alpha \not\models C] \le p(\bar{u}_C)^k$. Since C is a bad clause, $p(u_C) < \frac{1}{2}$, u_C is a negative literal $\bar{x}_C$, and $p(x_C) = \sqrt[k]{\frac{\mathrm{occ}_F(x_C)}{16e(F)}}$. We can calculate

$$\sum_{C\in\mathcal{B}} \Pr[\alpha \not\models C] \le \sum_{C\in\mathcal{B}} p(x_C)^k = \sum_{C\in\mathcal{B}} \frac{\mathrm{occ}_F(x_C)}{16e(F)} . \tag{3}$$

Since clause C contains $\bar{x}_C$, it conflicts with all $\mathrm{occ}_F(x_C)$ clauses containing x_C, thus $\sum_{C\in\mathcal{B}} \mathrm{occ}_F(x_C) \le 2e(F)$. The factor 2 arises since we count each conflict possibly twice—once from each side. Combining this with (3) proves the lemma. □

We cannot directly apply Lemma 1 to F. Therefore we apply the following sparsification process to F:

Lemma 5. *If F' does not contain the empty clause, then F is satisfiable.*

Proof. We will prove this using Lemma 1, the SAT version of the Lopsided Lovász Local Lemma. Fix a clause $C \in F'$. After the sparsification process, every literal u fulfills $\sum_{D:u\in D\in\mathcal{G}'} \Pr[\alpha \not\models D] \le \frac{1}{8k}$. We combine this with Lemma 4 to show

```
Algorithm: Sparsification Process
Let $\mathcal{G} = \{D \in F \mid p(u) \geq \frac{1}{2}, \forall u \in D\}$ be the set of good clauses in $F$.
$\mathcal{G}' := \mathcal{G}$
while $\exists$ a literal $u$ : $\sum_{D: u \in D \in \mathcal{G}'} \Pr[\alpha \not\models D] > \frac{1}{8k}$ do
    Let $C$ be some clause maximizing $\Pr[\alpha \not\models C]$ among all clauses in $\mathcal{G}'$
    containing $u$.
    $C' := C \setminus \{u\}$
    $\mathcal{G}' := (\mathcal{G}' \setminus \{C\}) \cup \{C'\}$
end
return $F' := \mathcal{G}' \cup \mathcal{B}$
```

that the condition (2) of the Local Lemma holds:

$$\begin{aligned}\sum_{D \in F':\ C \text{ and } D \text{ conflict}} \Pr[\alpha \not\models D] &= \sum_{D \in \mathcal{B}} \Pr[\alpha \not\models D] + \sum_{D \in \mathcal{G}':\ C \text{ and } D \text{ conflict}} \Pr[\alpha \not\models D] \\ &\leq \frac{1}{8} + \sum_{u \in C} \sum_{D \in \mathcal{G}' : \bar{u} \in D} \\ &\leq \frac{1}{8} + k \cdot \frac{1}{8k} = \frac{1}{4} \ .\end{aligned}$$

Hence (2) holds and by Lemma 1, F' is satisfiable, and clearly F as well. □

If F is unsatisfiable, the sparsification process produces the empty clause. We will show that in this case, $e(F)$ is large (at least $\Omega\left(2.69^k\right)$). If the sparsification process produces the empty clause, then there is some $C \in \mathcal{G}$ all whose literals are being deleted during the sparsification process. Write $C = \{u_1, u_2, \ldots, u_k\}$, and order the u_i such that $\mathrm{occ}_F(u_1) \leq \mathrm{occ}_F(u_2) \leq \cdots \leq \mathrm{occ}_F(u_k)$. Since C is a good clause, the definition of $p(x)$ implies that $p(u_1) \leq p(u_2) \leq \cdots \leq p(u_k)$. Fix any $\ell \in \{1, \ldots, k\}$ and let u_j be the first literal among $u_1, \ldots, u_\ell$ that is deleted from C. Let C' denote what is left of C just before that deletion, and consider the set $\mathcal{G}'$ at this point of time. Then $\{u_1, \ldots, u_\ell\} \subseteq C' \in \mathcal{G}'$. By the definition of the process,

$$\begin{aligned}\frac{1}{8k} &< \sum_{D:\ u_j \in D \in \mathcal{G}'} \Pr[\alpha \not\models D] \leq \sum_{D:\ u_j \in D \in \mathcal{G}'} \Pr[\alpha \not\models C'] \leq \\ &\leq \mathrm{occ}_F(u_j) \Pr[\alpha \not\models C'] \leq \\ &\leq \mathrm{occ}_F(u_\ell) \prod_{i=1}^{\ell} (1 - p(u_i)) \ .\end{aligned}$$

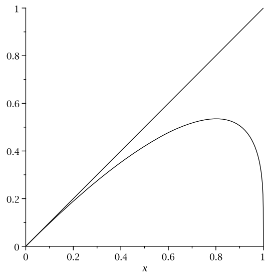
Since $p(u) \geq \sqrt[k]{\frac{\mathrm{occ}_F(u)}{16e(F)}}$ for all literals u in a good clause, it follows that $\frac{1}{128ke(F)} \leq p(u_\ell)^k \prod_{i=1}^{\ell}(1 - p(u_i))$, for every $1 \leq \ell \leq k$.

Let $(q_1, \ldots, q_k) \in [\frac{1}{2}, 1]^k$ be any sequence satisfying the k inequalities $\frac{1}{128ke(F)} \leq q_\ell^k \prod_{i=1}^{\ell}(1-q_i)$ for all $1 \leq \ell \leq k$. The $p(u_i)$ are such a sequence. We want to make the q_ℓ as small as possible: If $q_\ell > \frac{1}{2}$ and $\frac{1}{128ke(F)} < q_\ell^k \prod_{i=1}^{\ell}(1-q_i)$, we can decrease q_ℓ until one of the inequalities becomes an equality. The other $k-1$ inequalities stay satisfied. In the end we get a sequence $q_1, \ldots, q_k$ satisfying $\frac{1}{128ke(F)} = q_\ell^k \prod_{i=1}^{\ell}(1-q_i)$ whenever $q_\ell > \frac{1}{2}$. This sequence is non-decreasing: If $q_\ell > q_{\ell+1}$, then $q_\ell > \frac{1}{2}$, and $\frac{1}{128ke(F)} \leq q_{\ell+1}^k \prod_{i=1}^{\ell+1}(1-q_i) < q_\ell^k \prod_{i=1}^{\ell}(1-q_i) = \frac{1}{128ke(F)}$, a contradiction.

If all q_i are $\frac{1}{2}$, then the k^{th} inequality yields $128ke(F) \geq 4^k$, and we are done. Otherwise, there is some $\ell^* = \min\{i \mid q_i > \frac{1}{2}\}$. For $\ell^* \leq j < k$ both q_j and q_{j+1} are greater than $\frac{1}{2}$, thus $q_{j+1}^k \prod_{i=1}^{j+1}(1-q_i) = q_j^k \prod_{i=1}^{j}(1-q_i)$, and $q_j = q_{j+1}\sqrt[k]{1-q_{j+1}}$. We define

$$f_k(t) := t\sqrt[k]{1-t}\ ,$$

thus $q_j = f_k(q_{j+1})$. By $f_k^{(j)}(t)$ we denote $f_k(f_k(\ldots(f_k(t))\ldots))$, the j-fold iterated application of $f_k(t)$, with $f_k^{(0)}(t) = t$. In this notation, $q_j = f_k^{(k-j)}(q_k) > \frac{1}{2}$ for $\ell^* \leq j \leq k$. The figure below shows the graph of $f_4(t)$.



Proposition 2. *For $k \geq 2$ and any $t \in (0,1]$, $f_k^{(k-1)}(t) \leq \frac{1}{2}$.*

Please see the full version of the paper for the (easy) proof of this fact. By Proposition 2, $f_k^{(k-1)}(q_k) \leq \frac{1}{2}$, thus $\ell^* \geq 2$. Therefore $q_1 = \cdots = q_{\ell^*-1} = \frac{1}{2}$, and the $(l^*-1)^{\text{st}}$ inequality reads as

$$\frac{1}{128ke(F)} \leq q_{\ell^*-1}^k \prod_{i=1}^{\ell^*-1}(1-q_i) = 2^{-k-\ell^*+1}\ .$$

We obtain $e(F) \geq \frac{2^{k+\ell^*-1}}{128k}$. How large is ℓ^*? Define $S_k := \min\{\ell \in \mathbb{N}_0 \mid f_k^{(\ell)}(t) \leq \frac{1}{2}\ \forall t \in [0,1]\}$. By Proposition 2, $S_k \leq k-1$ and thus is finite. Since $f_k^{(k-\ell^*)}(q_1) = q_{\ell^*} > \frac{1}{2}$, we conclude that $k - \ell^* \leq S_k - 1$, thus $e(F) \geq \frac{2^{2k-S_k}}{128k}$.

Lemma 6. *The sequence $\frac{S_k}{k}$ converges to $\lim_{k\to\infty} \frac{S_k}{k} = -\int_{\frac{1}{2}}^{1} \frac{1}{x\ln(1-x)}dx < 0.572$.*

The proof of this lemma is technical and not related to satisfiability. Please see the full version of this paper for a proof. We conclude that $e(F) \geq \frac{2^{(2-0.572)k}}{128k} \in \Omega\left(2.69^k\right)$. $\square$

5 Conclusion

We want to give some hindsight why a sparsification procedure is necessary in both lower bound proofs in this paper. The probability distribution we define is not a uniform one, but biased towards setting x to `true` if $\text{occ}_F(x) \gg \text{occ}_F(\bar{x})$. The set of clauses conflicting with a specific clause C may contain many clauses containing some x with $\bar{x} \in C$. If x is the only literal in these clauses with $p(x) > \frac{1}{2}$, then each such clause is unsatisfied with probability not much smaller than 2^{-k}, and the sum (2) is greater than $\frac{1}{4}$ By removing x from these clauses, we reduce the number of clauses conflicting with C, making the sum (2) much smaller. However, for other clauses C', this sum might increase by removing x. We think that one will not be able to prove a tight lower bound using just a smarter sparsification process. We want to state some open problems and questions.

Question: Does $\lim_{k\to\infty} \sqrt[k]{gc(k)}$ exist?

If it does, it lies between 2.69 and 3.51. One way to prove existence would be to define "product" taking a k-CNF formula F and an ℓ-CNF formula G to a $(k+\ell)$-CNF formula $F \circ G$ that is unsatisfiable if F and G are, and $e(F \circ G) = e(F)e(G)$. With 2 and 4 ruled out, there seems to be no obvious guess for the value of the limit.

Question: Is there an $a > 2$ such that every unsatisfiable k-CNF formula contains a variable x with $\text{occ}_F(x) \cdot \text{occ}_F(\bar{x}) \geq a^k$?

Where do our methods fail to prove this? The part in the proof of the lower bound of Theorem 1 that fails is Lemma 4. On the other hand, Lemma 4 proves more than we need for Theorem 1: It proves that $\Pr[\alpha \models D]$, summed up over *all* bad clauses gives at most $\frac{1}{8}$. We only need that the bad clauses conflicting with a specific clause sum up to at most $\frac{1}{8}$. Still, we do not see how to apply or extend our methods to prove that such an $a > 2$ exists.

Acknowledgments. I am very thankful to Philipp Zumstein. We discussed the results of this paper extensively, and in particular the idea behind the proof of Lemma 6 is due to him.

References

1. Erdős, P., Spencer, J.: Lopsided Lovász Local Lemma and Latin transversals. Discrete Appl. Math. 30(2-3), 151–154 (1991); ARIDAM III, New Brunswick, NJ (1988)

2. Alon, N., Spencer, J.H.: The probabilistic method. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience (John Wiley & Sons), New York (2000); With an appendix on the life and work of Paul Erdős.
3. Lu, L., Székely, L.: Using Lovász Local Lemma in the space of random injections. Electron. J. Combin. 14(1), 13 (2007); Research Paper 63 (electronic)
4. Scheder, D., Zumstein, P.: How many conflicts does it need to be unsatisfiable? In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 246–256. Springer, Heidelberg (2008)
5. Erdős, P.: On a combinatorial problem. II. Acta Math. Acad. Sci. Hungar 15, 445–447 (1964)
6. Tovey, C.A.: A simplified NP-complete satisfiability problem. Discrete Appl. Math. 8(1), 85–89 (1984)
7. Kratochvíl, J., Savický, P., Tuza, Z.: One more occurrence of variables makes satisfiability jump from trivial to NP-complete. SIAM Journal of Computing 22(1), 203–210 (1993)
8. Savický, P., Sgall, J.: DNF tautologies with a limited number of occurrences of every variable. Theoret. Comput. Sci. 238(1-2), 495–498 (2000)
9. Hoory, S., Szeider, S.: A note on unsatisfiable k-CNF formulas with few occurences per variable. SIAM Journal on Discrete Mathematics 20(2), 523–528 (2006)
10. Gebauer, H.: Disproof of the neighborhood conjecture and its implications to sat (2008) (submitted)
11. Gebauer, H., Szabó, T., Tardos, G.: The local lemma is tight for SAT. In: Randall, D. (ed.) SODA, pp. 664–674. SIAM (2011)
12. Scheder, D., Zumstein, P.: How many conflicts does it need to be unsatisfiable? In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 246–256. Springer, Heidelberg (2008)
13. Erdős, P.: On a combinatorial problem. Nordisk Mat. Tidskr. 11, 5–10, 40 (1963)

Pursuit Evasion on Polyhedral Surfaces

Kyle Klein and Subhash Suri

University of California
Santa Barbara, CA, USA 93106
{kyleklein,suri}@cs.ucsb.edu

Abstract. We consider the following variant of a classical pursuit-evasion problem: *how many pursuers are needed to capture a single (adversarial) evader on the surface of a 3-dimensional polyhedral body?* The players remain on the closed polyhedral surface, have the same maximum speed, and are always aware of each others' current positions. This generalizes the classical lion-and-the-man game, originally proposed by Rado [12], in which the players are restricted to a two-dimensional circular arena. The extension to a polyhedral surface is both theoretically interesting and practically motivated by applications in robotics where the physical environment is often approximated as a polyhedral surface. We analyze the game under the discrete-time model, where the players take alternate turns, however, by choosing an appropriately small time step $t > 0$, one can approximate the continuous time setting to an arbitrary level of accuracy. Our main result is that 4 pursuers always suffice (upper bound), and that 3 are sometimes necessary (lower bound), for catching an adversarial evader on any polyhedral surface with genus zero. Generalizing this bound to surfaces of genus g, we prove the sufficiency of $(4g + 4)$ pursuers. Finally, we show that 4 pursuers also suffice under the "weighted region" constraints where the movement costs through different regions of the (genus zero) surface have (different) multiplicative weights.

1 Introduction

Pursuit-evasion problems serve as a mathematical abstraction for a number of applications that involve one group (pursuers) attempting to track down members of another group (evaders). Many such games with colorful names including Cops-and-Robbers, Hunter-and-Rabbit, Homicidal Chauffeur, and Princess-and-Monster have been studied in the literature [1,3,5,8]. We are inspired by the oldest such problem, the so-called *man-and-the-lion game*, in which a lion and a man are enclosed in a circular arena, both able to move continuously with the same maximum speed, and able to react instantaneously to each other's motion. Can the lion capture the man? For many years, it was believed that the following simple strategy guarantees a win for the lion in finite time: start at the center of the arena and continuously move toward the man along the radial line. This was proved false by Besicovitch who showed that the man can in fact evade the lion forever [12]: in Besicovitch's strategy, the lion can get arbitrarily close to the man but never quite reach it. This impossibility proof can be circumvented by either allowing the lion a fixed non-zero capture radius $r > 0$, or playing the game in discrete-time (alternating moves).

In this paper, we investigate the pursuit-evasion problem played on the (closed) surface of a 3-dimensional polyhedron. Multiple pursuers (lions) attempt to capture an adversarial evader (man), with all players constrained to remain on the polyhedral surface, and all able to move equally fast. In this setting, how many pursuers are needed to capture the evader in finite time? We study the problem in the discrete time model: this avoids the intractable problem of computing players' moves and reactions *instantaneously*, and also allows approximation of the continuous time setting to an arbitrary level of accuracy by choosing an appropriately small time step $t > 0$. On the practical side, the problem of pursuit on a polyhedral surface is well-motivated because many robotics applications involve searching or tracking on "terrain-like" surfaces. On the theoretical side, the problem is interesting because the surface acts as an "intrinsic" obstacle, introducing non-linearity in the behavior of shortest paths. For instance, *although the genus zero polyhedral surface is topologically equivalent to a disk, the game has a distinctly different character and outcome than its planar counterpart (circular arena).* In particular, it is known that a single pursuer can always win the discrete-time man-and-the-lion game in the plane (an easy corollary of [16]). Therefore, one may hope that an appropriate topological extension of the "follow the shortest path towards the evader" strategy will also succeed on the polyhedral surface. However, we show that this is not possible, and provide a constructive lower bound that *at least 3 pursuers* are needed in the worst-case for successful capture on a polyhedral surface. Intuitively, the problem is caused by the discontinuity in mapping "straight line" shortest paths in the unobstructed planar arena to geodesics on the polyhedral surface; in the unobstructed plane, a small move by the evader only causes a small (local) change in the straight line connecting pursuer and the evader, but on the polyhedral surface, the geodesic can jump discontinuously.

Complementing our lower bound, we show that 4 pursuers always suffice on any polyhedral surface of genus zero. Specifically, we present a strategy for the pursuers that always leads to capture of the evader in $O(\Delta_S(n^2 \log n + \log \Delta_S))$ time steps, where n is the number of vertices of the polyhedral surface S and Δ_S is its diameter (the maximum shortest path distance between any two points). We then generalize our result to surfaces of non-zero genus and prove that $(4+4g)$ pursuers can always capture an evader on the surface of any genus g polyhedron. Our technique for analyzing pursuit evasion on polyhedral surfaces appears to be quite general, and likely to find application in other settings. As one example, we consider pursuit evasion under the "weighted region" model of shortest paths, where non-negative weights dictate the per-unit cost of travel through different regions of the surface.

Related Work

In the discrete-time model, a single pursuer can capture the evader in a simply-connected polygon [7], while 3 pursuers are both necessary and sufficient for polygonal environments with multiple holes (obstacles) [4]. In a visibility-sensing model, where pursuers can localize the evader only when the latter is in direct line of sight, the number of pursuers is $O(\sqrt{h} + \log n)$ for n-vertex environment with h holes [9].

There exists an extensive literature on pursuit-evasion in 3-dimensional environments and surfaces, but no result appears to be known on the number of pursuers necessary for capture. Instead, the prior research has focussed on heuristics approaches for capture [10], classification of environments where capture is achievable [2], or on game-theoretical questions [11,13].

The most relevant work to our research is the cops-and-robbers games in graph theory, where Aigner and Fromme have shown that 3 cops always suffice against a robber in any planar graph [1], and $\lfloor 3g/2 + 3 \rfloor$ cops are necessary for graphs of genus g [15]. However, the continuous-space of polyhedral surfaces requires very different set of techniques from those used for graphs.

2 Preliminaries and the Lower Bound

The geometric environment for our pursuit-evasion problem is the (closed) surface of a 3-dimensional polyhedron S. We assume that S has n vertices, and therefore $O(n)$ faces and edges. Without loss of generality, we assume that each face is a triangle, which is easily achieved by triangulating the faces with four or more sides. We use the notation $p_1, p_2, \ldots$ to denote the group of pursuers who wish to track and capture a single (adversarial) evader e. Slightly abusing the notation, we also use e and p_i, respectively, for the current location of the evader and the ith pursuer.

We make the standard assumption about the game: all the players know the environment (the surface of the polyhedron S), each player knows the current positions of all the other players, all players have identical maximum speed, and the game is played in the discrete-time alternating turn model. By an appropriate scaling of the environment, we assume that the maximum speed of the players is 1, meaning that on its turn a player can move to any position within *geodesic distance* one of its current location on the surface. On their turn, all the pursuers move simultaneously. The pursuers win the game if, on their turn, some p_i reaches the current position of the evader and the evader wins if it can avoid capture indefinitely.

We use the notation $P_{a,b}$ for a shortest path between two points a and b on the surface S, and $d(a, b)$ for the length of this path. (In general, the path $P_{a,b}$ is not unique, but its length is.) The path $P_{a,b}$ is piece-wise linear and its vertices lie on the edges or vertices of the surfaces. Throughout, we will use the terms *vertices* and *edges* to refer to the graph of the polyhedral surface, and *points* and *arcs* to refer to the geometric objects embedded on the surface such as a path. We explain specific properties of these shortest paths that are used in our analysis in Section 3.3. The following theorem establishes the lower bound for our pursuit game.

Theorem 1. *In the worst-case at least three pursuers are required to capture an evader on the surface of a polyhedron.*

Proof. We start with a dodecahedron D, all of whose edges have length 1 (see Fig. 1(a). Our polyhedron S is constructed by extending each face of D *orthogonally* (to the face) into a "tower" of height $\Delta_D + 1$, where Δ_D is geodesic diameter of the dodecahedron; see Fig. 1(b). S has 12 such towers, one for each of the 12 pentagonal faces of D. The "walls" of these towers meet along the edges of D, forming the skeleton graph, which we denote $G(D)$, as shown in Fig. 1(c). We argue that an evader can indefinitely avoid capture from two pursuers on the surface of this polyhedron. In particular, the two pursuers, p_1 and p_2 initially choose their positions, and then the evader picks its initial position at a vertex of $G(D)$ to satisfy $d(p_i, e) > 1$, for $i = 1, 2$. We show that regardless of the pursuers' strategies, the evader can indefinitely maintain this distance condition (after its move) by always moving among the vertices of $G(D)$. The evader's strategy is *reactive*: it remains at a vertex until some pursuer is within distance 1. When one or both pursuers are within distance 1 of the evader, we show that the evader can move to a safe neighboring vertex and restore its distance condition. Due to space limitation, we omit the further details and refer the reader to the full version of the paper.

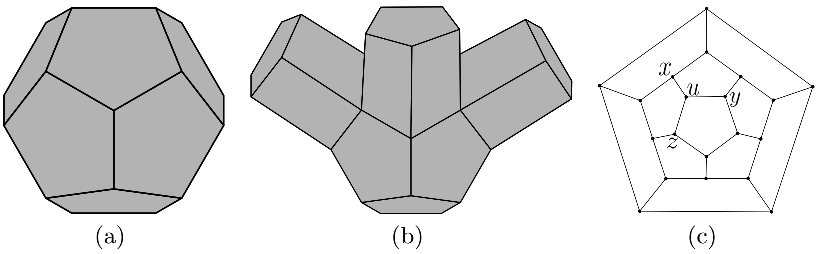


Fig. 1. A dodecahedron (a); partial construction with three faces orthogonally extended (b); and the skeleton graph (c)

3 Catching the Evader with 4 Pursuers

We begin with a high level description of the pursuers' strategy, and then develop the necessary technical machinery to prove its correctness.

3.1 Surround-and-Contract Pursuit Strategy

The pursuers' overall strategy is conceptually quite simple: repeatedly shrink the region containing the evader while making sure that it cannot escape from this region, which can be intuitively thought of as a *surround-and-contract* strategy. More specifically, at any time, the evader is constrained within a connected portion S_i of the surface S, which is bounded by at most three paths, each guarded by a pursuer. The fourth pursuer is used to divide S_i into two non-empty regions (contraction), trapping the evader within one of them. This division is done in such a way that that at least one of the 3 pursuers bounding S_i becomes free, thus allowing the process to continue until the target region reduces to a single triangle, and the capture can be completed.

The paths used by the pursuers are shortest paths on the polyhedral surface, *restricted to the current region.* The computation of shortest paths on a polyhedral surface is a well-known problem in computational geometry, and we rely on the following result of [6,14]: given a source point x on the surface of a polyhedron S of n vertices, one can compute a shortest path map encoding the shortest paths from x to all other points on S, in $O(n^2)$ time using $O(n \log n)$ space. With this map, one can find the shortest path from x to any other point y in time $O(\log n + k)$ when the path consists of k arcs.

We use *phases* to monitor the progress of the algorithm: in phase i, the region containing the evader is denoted S_i where $S_i \subseteq S_{i-1}$, for all i. Each time the pursuers guard a new path dividing S_i, the phase transitions, with S_{i+1} as the region containing the evader. In addition, each region S_i has a rather special form: it is bounded by either two or three shortest paths. The finite automaton of Figure 2 shows the simple state diagram of the pursuit: the pursuit transitions between regions bounded by 2 and 3 paths until it reaches a special terminal state marked EndGame. For ease of reference, we name the first two states BiPolar and TriPolar to emphasize that the regions corresponding to these states are bounded by shortest paths between 2 or 3 points (poles). The region in the terminal state EndGame is also bounded by 3 paths but contains no vertices in the interior (only the points of the boundary paths), which simplifies the search leading to capture. In particular, the three possible states throughout the pursuit are the following:

BiPolar: S_i is bounded by two shortest paths $P_{a,b}$ and $P'_{a,b}$ between two points (poles) a and b.

TriPolar: S_i contains at least one interior vertex, and is bounded by three shortest paths $P_{a,b}$, $P_{b,c}$, and $P_{a,c}$.

EndGame: S_i has no interior vertices and is bounded by three shortest paths $P_{a,b}$, $P_{b,c}$, and $P_{a,c}$.

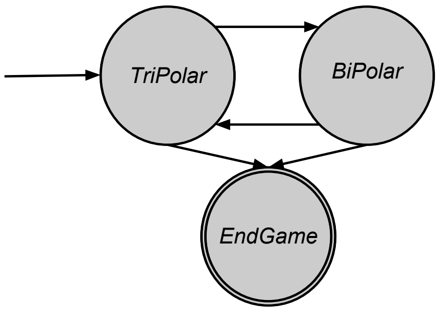


Fig. 2. A finite state machine representing the possible states of the pursuit and transitions between them

We initialize the pursuit by choosing a triangular face (a, b, c) of the surface, and assigning one pursuer to each of the three (single-arc) shortest paths $P_{a,b}$, $P_{b,c}$, and $P_{a,c}$. If the evader lies inside the triangle face, we enter the terminal state EndGame; otherwise, we are in state TriPolar. The fourth pursuer shrinks the region S_i, resulting in a smaller TriPolar region, or forces a transition to a BiPolar region. In each state BiPolar, at least one interior vertex is eliminated from S_i. Further, each state consists of a finite number of phases, which guarantees that the algorithm terminates in the region EndGame.

In the following, we use $\nu(S_i)$ to denote the *number of interior* vertices of S_i; that is, the number of vertices in S_i that are not on the boundary paths. Throughout the pursuit, the following invariant is maintained.

Pursuit Invariant. During the ith phase of the pursuit, (1) $S_i \subseteq S_{i-1}$, (2) $\nu(S_i) \leq \nu(S_{i-1})$, and if phase $i-1$ is in state BiPolar, then $\nu(S_i) < \nu(S_{i-1})$, and (3) at most 4 paths are guarded, each by a single pursuer at any time.

The first condition ensures that the region containing the evader only shrinks; the second ensures that at least one interior vertex is removed in state BiPolar; and the third ensures that 4 pursuers succeed in capturing the evader.

3.2 Guarding Shortest Paths

Our algorithm employs one pursuer to guard a shortest path, ensuring that any attempt by the evader to cross the shortest path leads to capture. The key idea behind this strategy is the "projection" of the evader along the shortest path, defined as follows.

Projection. Given a shortest path $P_{a,b}$ between two points a and b, and the current evader location e, a point e_π on $P_{a,b}$ is called the *projection* of e if $d(e_\pi, x) \leq d(e, x)$, for all $x \in P_{a,b}$.

That is, if a pursuer p is positioned at e_π, then it is always closer than evader to every point of $P_{a,b}$, and therefore any move by the evader crossing $P_{a,b}$ leads to capture by p on its next move. While multiple projections may exist, the pursuers will guard a path by maintaining their location at the *canonical projection* of the evader, defined as follows.

Canonical Projection. Given a shortest path $P_{a,b}$ between two points a and b, and the current evader location e, a point e_π on $P_{a,b}$ is called the *canonical projection* of e if $d(a, e_\pi) = min(d(a, e), d(a, b))$.

The following three lemmas establish the technical preliminaries about the existence, maintainability, and reachability of the canonical projection. Due to space limitation, we omit the further details and refer the reader to the full version of the paper. *Throughout, a shortest path always means the minimum length path restricted to the current subsurface S_i, and e_π refers to the unique canonical projection.*

Lemma 1. *Given any shortest path $P_{a,b}$ on the polyhedral surface, the canonical projection e_π is a projection of the evader.*

Lemma 2. *Suppose the current position of the evader is e, the pursuer p is positioned at the canonical projection e_π on the shortest path $P_{a,b}$, and the evader moves to a new position e'. Then, p can reposition itself at the new canonical projection e'_π in one move, or capture the evader if the evader's move crossed the path $P_{a,b}$.*

Lemma 3. *Consider a shortest path $P_{a,b}$ on the polyhedral surface S, and suppose a pursuer p is located at the endpoint a of this path. Then, after at most $L + 1$ moves, p can locate itself at the canonical projection of the evader, where L is the (Euclidean) length of the $P_{a,b}$.*

These lemmas together show that a single pursuer is able to guard a shortest path on the surface. We now describe the pursuers' strategy for each of the three states: BiPolar, TriPolar, EndGame.

3.3 Pursuit Strategy for the TriPolar State

In TriPolar state, the current region S_i is bounded by three shortest paths, $P_{a,b}$, $P_{a,c}$, and $P_{b,c}$, between the three poles a, b, c. The pursuers' strategy is to force the game either into BiPolar or EndGame state while preserving the Pursuit Invariant. Towards that goal, we need to introduce some properties of shortest paths on polyhedral surfaces.

It is well-known that a shortest path is a sequence of line segments (arcs), whose endpoints lie on the edges of the surface, and that the path crosses any edge of the surface at most once. Thus, the sequence of edges crossed by a path, called the *edge sequence*, consists of at most n edges. Given a source point a and an edge (b, c), it is also known that (b, c) is partitioned into $O(n)$ closed *intervals of optimality* [14], where the shortest path from a to any point d in an interval follows the same edge sequence. Let us suppose that an edge (b, c) is partitioned into k intervals of optimality, $[d_0, d_1], [d_1, d_2], \cdots, [d_{k-1}, d_k]$, where the edge sequence for the interval $[d_{i-1}, d_i]$ is denoted as σ_i. Since two adjacent intervals, say $[d_{j-1}, d_j]$ and $[d_j, d_{j+1}]$, share a common endpoint d_j, there are two equal length shortest paths from a to d_j, following edge sequences σ_j and σ_{j+1}. Because our algorithm may guard one or both of these shortest paths, we use a superscript to identify the associated edge sequence. In particular, the shortest path from x to y under the edge sequence σ_j is denoted $P^j_{x,y}$.

The following lemma shows that if the shortest paths $P_{a,b}$ and $P_{a,c}$ have the same edge sequence, and $P_{b,c}$ is a single arc, then the interior of the region bounded by these 3 paths has no vertex of the surface, which implies that the pursuit region has entered the terminal state EndGame.

Lemma 4. *Suppose the current region S_i is bounded by pairwise shortest paths between the three points a, b, c, and that $P_{b,c}$ consists of a single arc. Then, the paths $P_{a,b}$ and $P_{a,c}$ follow the same edge sequence if and only if S_i contains no interior vertices.*

Proof. Clearly, if $P_{a,b}$ and $P_{a,c}$ have the same edge sequence, then there cannot be an interior vertex in S_i because $P_{b,c}$ is a single arc. For the converse, if S_i has no interior vertices and $P_{b,c}$ is a single arc, then S_i can only contain edges that intersect both $P_{a,b}$ and $P_{a,c}$. These edges do not cross each other, and therefore they must be crossed by $P_{a,b}$ and $P_{a,c}$ in the same order. □

By the preceding lemma, if $P_{a,b}$ and $P_{a,c}$ follow the same edge sequence and $P_{b,c}$ consists of a single arc, then we are in the terminal state EndGame. Therefore, assume that either the edge sequences of $P_{a,b}$ and $P_{a,c}$ are unequal or $P_{b,c}$ consists of multiple arcs. In both cases, the following lemma shows how to either reduce $P_{b,c}$ to a single point, which changes the state to BiPolar, or replace $P_{a,b}$ and $P_{a,c}$ with shortest paths with the same edge sequence, and $P_{b,c}$ with a single arc, which changes the state to EndGame.

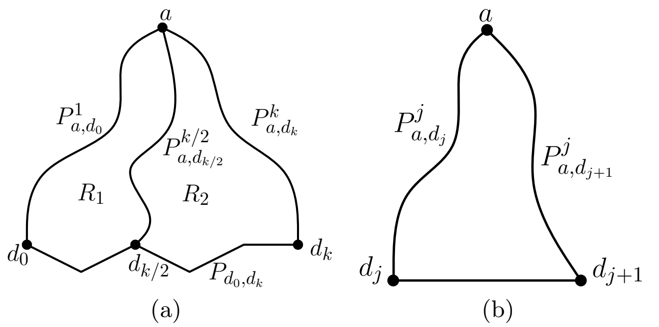


Fig. 3. Illustration for the proof of Lemma 5

Lemma 5. *Suppose S_i is in state* TriPolar*, then we can force a transition either to state* BiPolar *or state* EndGame*.*

Proof. Consider the shortest path map with source a, and suppose it partitions $P_{b,c}$ into k intervals of optimality (across all of $P_{b,c}$'s arcs), $[d_0, d_1]$, $[d_1, d_2] \cdots$, $[d_{k-1}, d_k]$ with corresponding edge sequences $\sigma_1, \sigma_2, \cdots, \sigma_k$, where $d_o = b$ and $d_k = c$. Relabel $P_{a,b}$ as P^1_{a,d_0}, and $P_{a,c}$ as P^k_{a,d_k}, and order the paths by their endpoints on $P_{b,c}$ as follows:

$$P^1_{a,d_0}, P^1_{a,d_1}, P^2_{a,d_1}, P^2_{a,d_2}, \ldots, P^k_{a,d_{k-1}}, P^k_{a,d_k}$$

We leave two pursuers to guard (maintain canonical projections on) the paths $P_{a,b}$ and $P_{a,c}$, and deploy a guard on the center path $P^{k/2}_{a,d_{k/2}}$ (constrained to lie within the current region); see Figure 3(a). This path splits the original region S_i into two non-empty regions, each containing half the intervals of optimality, and we recurse the process on the side with the evader, namely, the region S_{i+1}. The first two conditions of the invariant are trivially satisfied, since the evader region can only shrink, and the third condition holds because the pursuer associated with either the path $P_{a,b}$ or $P_{a,c}$ is freed up, keeping the total pursuer count at four.

The recursion terminates when the evader is confined between two successive paths in the original ordering. In particular, if the evader is trapped between paths P^j_{a,d_j} and $P^j_{a,d_{j+1}}$, then we have state EndGame as shown shown in Fig. 3(b). On the other hand, if the evader is trapped between two paths P^{j-1}_{a,d_j} and P^{j}_{a,d_j}, we have successfully transitioned to state BiPolar, as shown in Fig. 3(c). It is clear that throughout this search, the evader remains confined to a subsurface of S_i and cannot escape without being captured, and that the pursuit invariant is maintained. Because the path $P_{b,c}$ has at most n arcs, with n intervals of optimality each, we have $k \le n^2$. Thus, in $O(\log n)$ phases, we can force a change of state to either BiPolar or EndGame. □

3.4 Pursuit Strategy for the BiPolar State

We now describe how to make progress when the search region is BiPolar. Without loss of generality, assume that the current region S_i is bounded by two shortest paths between points a and b, each guarded by a pursuer. The algorithm shrinks the region by removing at least one vertex from the interior of S_i. In particular, let c be a vertex of the surface that lies in the interior, and consider the two shortest paths (constrained to remain inside S_i) from c to a and b. The concatenation of these two paths splits S_i into two subregions, say R_1 and R_2, both bounded by three paths. (These paths can share a common prefix, starting at c, but they do not cross each other.) Only one of these regions contains the evader, and so by guarding $P_{a,b}$ an $P_{a,c}$ the state of the search transitions to either TriPolar or EndGame depending on whether or not this region, which becomes S_{i+1}, contains an interior vertex. See Figure 4 for illustration. During this transition the pursuit invariant holds because (1) $R_1, R_2 \subseteq S_i$, (2) both R_1 and R_2 contain at least one fewer interior vertex, namely, c, and (3) at most 4 pursers are used. Thus, we have established the following lemma, completing the discussion of the state BiPolar.

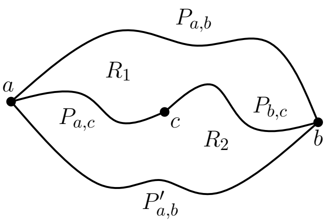


Fig. 4. An abstract illustration of the two paths that may be guarded during state BiPolar

Lemma 6. *If the evader lies in a* BiPolar *region S_i, then we can force a transition to a* TriPolar *or* EndGame *region with at least one fewer interior vertex, and no more than 4 pursuers are used during the pursuit.*

3.5 Pursuit Strategy for the EndGame State

We now describe how the pursuers capture the evader when the search region is EndGame. First, by Lemma 5, the path $P_{b,c}$ can be reduced to a single arc. Next, by Lemma 4, since S_i has no interior vertices, $P_{a,b}$ and $P_{a,c}$ follow the same edge sequence. Thus, S_i consists of a chain of faces, each a triangle or a quadrilateral. For ease of presentation, we assume that all faces are triangles, which is easily achieved by adding a diagonal to each quadrilateral. The pursuers perform a sweep of S_i, by repeatedly replacing $P_{b,c}$ with the previous edge in the edge sequence of $P_{a,b}$ and $P_{a,c}$, until the evader is trapped in a triangle each of whose sides are guarded by a pursuer. For example, in Figure 5(a), the fourth pursuer guards the edge (b, x_1), which either confines the evader to the triangle b, c, x_1 or frees the evader guarding $P_{b,c}$.

Lemma 7. *Once the evader enters the* EndGame *state, the 4 pursuers can shrink the confinement region to a single triangle of S_i in $O(n)$ phases.*

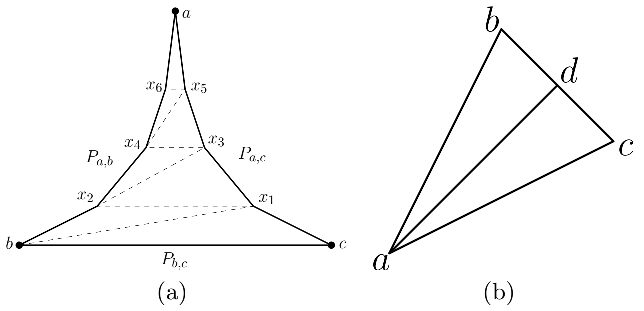


Fig. 5. Illustrating the algorithm used for capture in state EndGame

Finally, the following lemma completes the capture inside the triangle.

Lemma 8. *If S_i consists of a single triangle, then in $O(\Delta_S \log \Delta_S)$ moves the evader can be captured.*

Proof. The pursuers progressively "shrink" the triangle containing the evader, leading to eventual capture, as follows. Pick the midpoint of the arc (b, c), say d, and deploy a guard on the arc (a, d); see Figure 5(b). This path splits the original triangle into two non-empty triangles, and we recurse the process on the triangle containing the evader. Notice that the pursuer associated with either the path $P_{a,c}$ or $P_{a,b}$ is freed up, keeping the total pursuer count at four. After $\log \Delta_S$ applications (b, c) will be replaced with an arc of length at most one, at which point a pursuer can capture the evader by sweeping the triangle once. At most $O(\log \Delta_S)$ paths of length $O(\Delta_S)$ are guarded, and so this process takes at most $O(\Delta_S \log \Delta_S)$ moves. □

We can now state our main result.

Theorem 2. *On a n-vertex genus 0 polyhedral surface S, 4 pursuers can always capture the evader in $O(\Delta_S(n^2 \log n + \log \Delta_S))$ moves.*

4 Extensions and Generalizations

Our surround-and-contract technique appears to be quite general, and may be applicable to many other settings where shortest paths are well-behaved and where the frequency of state transitions between BiPolar and TriPolar can be combinatorially bounded. In particular, we have the following two results, whose details can be found in the full version of the paper.

Theorem 3. *On a n-vertex genus g polyhedral surface S, 4g + 4 pursuers can always capture the evader in* $O(((gn)^2 \log(gn) + \log \Delta_S) \cdot \Delta_S)$ *moves.*

Theorem 4. *Given a polyhedron S with n vertices, and weighted regions with min weight* ω_{min} *and max weight* ω_{max}*, 4 pursuers can capture the evader in* $O(\frac{\omega_{max}}{\omega_{min}} \cdot n^6 \cdot \Delta_S + \log((\frac{\omega_{max}}{\omega_{min}}) \cdot \Delta_S) \cdot \frac{\omega_{max}}{\omega_{min}} \cdot \Delta_S)$ *moves.*

References

1. Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics 8(1), 1–12 (1984)
2. Alexander, S., Bishop, R., Ghrist, R.: Capture pursuit games on unbounded domains. Enseign. Math. (2) 55(1-2), 103–125 (2009)
3. Alpern, S., Fokkink, R., Lindelauf, R., Olsder, G.-J.: The "princess and monster" game on an interval. SIAM J. on Control and Optimization 47(3), 1178–1190 (2008)
4. Bhadauria, D., Klein, K., Isler, V., Suri, S.: Capturing an evader in polygonal environments with obstacles: The full visibility case. International Journal of Robotics Research 31(10), 1176–1189 (2012)
5. Bopardikar, S.D., Bullo, F., Hespanha, J.: A cooperative homicidal chauffeur game. In: 46th IEEE Conference on Decision and Control, pp. 4857–4862 (2007)
6. Chen, J., Han, Y.: Shortest paths on a polyhedron. In: Proc. of 6th Symposium on Computational Geometry, pp. 360–369. ACM, New York (1990)
7. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion in a polygonal environment. IEEE Transactions on Robotics 21(5), 875–884 (2005)
8. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion with local visibility. SIAM Journal on Discrete Mathematics 1, 26–41 (2006)
9. Klein, K., Suri, S.: Catch me if you can: Pursuit and capture in polygonal environments with obstacles. In: Proc. of 26th Conference on Artificial Intelligence, pp. 2010–2016 (2012)
10. Kolling, A., Kleiner, A., Lewis, M., Sycara, K.: Pursuit-evasion in 2.5d based on team-visibility. In: Proc. IROS, pp. 4610–4616 (2010)
11. Kovshov, A.: The simple pursuit by a few objects on the multidimensional sphere. In: Game Theory & Applications II, pp. 27–36 (1996)
12. Littlewood, J.E.: Littlewood's Miscellany. Cambridge University Press (1986)
13. Melikyan, A.: Geometry of pursuit-evasion games on two-dimensional manifolds. Annals of the International Society of Dynamic Games 9, 173–194 (2007)
14. Mitchell, J.S.B., Mount, D.M., Papadimitriou, C.H.: The discrete geodesic problem. SIAM Journal on Computing 16(4), 647–668 (1987)
15. Schroder, B.S.W.: The copnumber of a graph is bounded by $\lfloor$ 3/2 genus(g)$\rfloor$ +3. In: "Categorical Perspectives" — Proc. of the Conference in Honor of George Strecker's 60th Birthday, pp. 243–263 (2001)
16. Sgall, J.: Solution of david gale's lion and man problem. Theor. Comput. Sci. 259(1-2), 663–670 (2001)

Algorithms for Tolerated Tverberg Partitions

Wolfgang Mulzer and Yannik Stein*

Institut für Informatik, Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
{mulzer,yannikstein}@inf.fu-berlin.de

Abstract. Let P be a d-dimensional n-point set. A partition $\mathcal{T}$ of P is called a *Tverberg* partition if the convex hulls of all sets in $\mathcal{T}$ intersect in at least one point. We say $\mathcal{T}$ is *t-tolerated* if it remains a Tverberg partition after deleting any t points from P. Soberón and Strausz proved that there is always a t-tolerated Tverberg partition with $\lceil n/(d+1)(t+1)\rceil$ sets. However, so far no nontrivial algorithms for computing or approximating such partitions have been presented.

For $d \leq 2$, we show that the Soberón-Strausz bound can be improved, and we show how the corresponding partitions can be found in polynomial time. For $d \geq 3$, we give the first polynomial-time approximation algorithm by presenting a reduction to the (untolerated) Tverberg problem. Finally, we show that it is coNP-complete to determine whether a given Tverberg partition is t-tolerated.

1 Introduction

Let $P \subset \mathbb{R}^d$ be a point set of size n. A point $c \in \mathbb{R}^d$ has *(Tukey) depth m* with respect to P if every closed half-space containing c also contains at least m points from P. A point of depth $\lceil n/(d+1)\rceil$ is called a *centerpoint* for P. The well-known Centerpoint Theorem [10] states that any point set has a centerpoint. Centerpoints are of great interest as they constitute a natural generalization of the median to higher-dimensions and since they are invariant under scaling or translations and robust against outliers.

Chan [1] described a randomized algorithm that finds a d-dimensional centerpoint in expected time $\mathcal{O}(n^{d-1})$. Actually, Chan solves the seemingly harder problem of finding a point with maximum depth, and he conjectures that his result is optimal. Since this is infeasible in higher dimensions, approximation algorithms are of interest. Already in 1993, Clarkson et al. [2] developed a Monte-Carlo algorithm that finds a point with depth $\Omega(n/(d+1)^2)$ in time $\mathcal{O}(d^2(d\log n+\log(1/\delta))^{\log(d+2)})$, where δ is the error-probability. Teng [13] proved that testing whether a given point is a centerpoint is coNP-complete, so we do not know how to verify efficiently the output of the algorithm by Clarkson et al. For a subset of centerpoints, *Tverberg partitions* [14] provide polynomial-time checkable proofs for the depth: a *Tverberg m-partition* for a point set $P \subset \mathbb{R}^d$ is

* This research was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

a partition $P = T_1 \dot\cup T_2 \dot\cup \dots \dot\cup T_m$ of P into m sets such that $\bigcap_{i=1}^{m} \mathsf{conv}(T_i) \neq \emptyset$. Each half-space that intersects $\bigcap_{i=1}^{m} \mathsf{conv}(T_i)$ must contain at least one point from each T_i, so each point in $\bigcap_{i=1}^{m} \mathsf{conv}(T_i)$ has depth at least m. Tverberg's theorem states that $m = \lceil n/(d+1) \rceil$ is always possible. Thus, there is always a centerpoint with a corresponding Tverberg partition. Miller and Sheehy [7] developed a deterministic algorithm that computes a point of depth $\lceil n/2(d+1)^2 \rceil$ in time $n^{\mathcal{O}(\log d)}$ together with a corresponding Tverberg partition. This was recently improved by Mulzer and Werner [9]. Through recursion on the dimension, they can find a point of depth $\lceil n/4(d+1)^3 \rceil$ and a corresponding Tverberg partition in time $d^{\mathcal{O}(\log d)} n$.

Let $\mathcal{T}$ be a Tverberg m-partition for P. If any nonempty subset $R \subset P$ is removed from P, we do not longer know if $\bigcap_{i=1}^{m} \mathsf{conv}(T_i \setminus R) \neq \emptyset$. In the worst-case, the maximum number of sets in $\mathcal{T}$ whose convex hulls still have a nonempty intersection is $m - |R|$. This is not always desired. It is therefore of interest to study Tverberg partitions that guarantee $\bigcap_{i=1}^{m} \mathsf{conv}(T_i \setminus R)$ to be nonempty if the size of R is not "too big". We call a Tverberg partition *t-tolerated* if it remains a Tverberg partition of P even after removing t arbitrary points from P. In 1972, Larman [5] proved that every set of size $2d+3$ admits a 1-tolerated Tverberg 2-partition. This was motivated by a problem that was proposed to him by McMullen: find the largest number of points that can be brought in convex position by a permissible projective transformation. Colín [4] generalized Larman's result, showing that sets of size $(t+1)(d+1)+1$ always have a t-tolerated Tverberg 2-partition. Later, Montejano and Oliveros conjectured that every set of size $(t+1)(m-1)(d+1)+1$ admits a t-tolerated Tverberg m-partition [8, Conjecture 4.2]. This was proven by Soberón and Strausz [12] who adapted Sarkaria's proof of Tverberg's theorem [11] to the tolerated setting. Soberón and Strausz also conjectured this bound to be tight [12, Conjecture 1]. Up to now, no exact or approximation algorithms for tolerated Tverberg partitions appear in the literature.

In this paper, we give new bounds for one- and two-dimensional tolerated Tverberg partitions, disproving the Soberón-Strausz-conjecture. We also give efficient algorithms for finding the corresponding partitions. Our bound is tight for $d = 1$. For higher dimensions, we describe an approximation preserving reduction to the untolerated Tverberg problem. Thus, we can apply existing and possible future algorithms for the untolerated Tverberg problem in the tolerated setting. Finally, we show that testing whether a given Tverberg partition has tolerance t is coNP-complete if the dimension is not fixed.

2 Low Dimensions

We start with an algorithm for the one-dimensional case that yields a tight bound. This can be bootstrapped to higher dimensions with a lifting approach similar to [9]. In two dimensions, we also get an improved bound if the size of the desired partition and the tolerance is large enough.

2.1 One Dimension

Let $P \subset \mathbb{R}$ with $|P| = n$, and let $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ be a t-tolerated Tverberg m-partition of P. By definition, there is no subset $R \subset P$, $|R| = t$ whose removal separates the convex hulls of the sets in $\mathcal{T}$. Bounding the size of the sets in $\mathcal{T}$ gives us more insight into the structure.

Lemma 2.1. *Let $P \subset \mathbb{R}$ with $|P| = n$ and $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ a t-tolerated Tverberg m-partition of P. Then*

(i) for $i = 1, \ldots, m$, we have $|T_i| \geq t + 1$; and
(ii) for $i, j = 1, \ldots, m$, $i \neq j$, we have $|T_i \cup T_j| \geq 2t + 3$.

Proof. (i) Suppose $|T_i| \leq t$. After removing T_i from P, the intersection of the convex hulls of the sets in $\mathcal{T}$ becomes empty, and $\mathcal{T}$ would not be t-tolerated.
(ii) Suppose there are $T_i, T_j \in \mathcal{T}$ with $|T_i \cup T_j| \leq 2t + 2$. By (i), we have $|T_i| = |T_j| = t+1$. Let $p_{\min} = \min(T_i \cup T_j)$ and assume w.l.o.g. that $p_{\min} \in T_i$ (see Figure 1). Then $|T_i \setminus \{p_{\min}\}| = t$, and removing the set $T_i \setminus \{p_{\min}\}$ separates the convex hulls of T_i and T_j. This again contradicts $\mathcal{T}$ being t-tolerated.

□

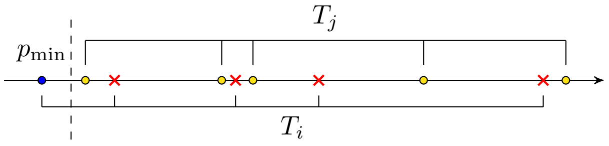


Fig. 1. The convex hulls of two sets of size $t+1$ can be separated by removing t points

Lemma 2.1 immediately implies a lower bound on the size of any point set that admits a t-tolerated Tverberg m-partition.

Corollary 2.2. *Let $P \subset \mathbb{R}$ with $|P| < m(t+2) - 1$. Then P has no t-tolerated Tverberg m-partition.*

Now what happens for $|P| = m(t+2) - 1$? Note that for $t > 0$ and $m > 2$, we have $m(t+2) - 1 < 2(t+1)(m-1) + 1$, the bound by Soberón and Strausz. Thus, proving that a t-tolerated Tverberg m-partition exists for any one-dimensional point set of size $m(t+2) - 1$ would disprove the Soberón-Strausz conjecture.

Let $P \subset \mathbb{R}$ be of size $m(t+2) - 1$. By Lemma 2.1, in any t-tolerated Tverberg partition of P, one set has to be of size $t+1$ and all other sets have to be of size $t+2$. Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be a Tverberg m-partition of P such that T_1 contains every mth point of P and each other set T_i $(i \geq 2)$ has one point in each interval defined by the points of T_1; see Fig. 2 for $m = 3$ and $t = 2$. Note that $|T_1| = t+1$ and $|T_i| = t+2$ $(i \geq 2)$. We will show that $\mathcal{T}$ is t-tolerated. Intuitively, $\mathcal{T}$ maximizes the interleaving of the sets, making the convex hulls more robust to changes.

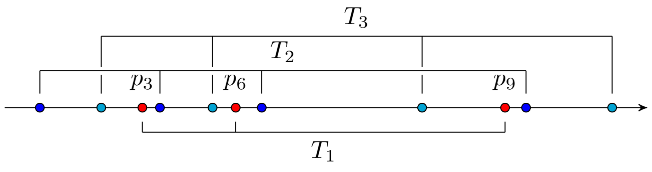


Fig. 2. A 2-tolerated Tverberg 3-partition for 11 (= 3(2 + 2) − 1) points

Lemma 2.3. *Let $P \subset \mathbb{R}$ with $|P| = m(t+2)-1$, and let $\mathcal{T} = \{T_1, \ldots, T_m\}$ be an m-partition of P. Suppose that $|T_1| = t+1$, and write $T_1 = (p_1, p_2, \ldots, p_{t+1})$, sorted from left to right. Suppose that each interval $\mathcal{I} \in \{(-\infty, p_1),\ (p_1, p_2),\ \ldots,(p_{t+1}, \infty)\}$ contains one point from each T_i, for $i = 2, \ldots, m$. Then $\mathcal{T}$ is a t-tolerated Tverberg m-partition for P.*

Proof. Suppose there exist $T_i, T_j \in \mathcal{T}$, $(i \neq j)$ and a subset $R \subset P$ of size t such that removing R from P separates the convex hulls of T_i and T_j. Let h be a point that separates $\mathsf{conv}(T_i \setminus R)$ and $\mathsf{conv}(T_j \setminus R)$. Let $T_i^- = T_i \cap (-\infty, h]$ and $T_i^+ = T_i \cap (h, \infty)$, and define T_j^-, T_j^+ similarly. Figure 3 shows the situation. Set $l = |T_1^-| = |T_1 \cap (-\infty, h]|$. By construction of $\mathcal{T}$, both T_i^- and T_j^- contain exactly l or $l+1$ points.

Since removing R separates the convex hulls of T_i and T_j at h, R must contain either $T_i^- \cup T_j^+$ or $T_j^- \cup T_i^+$. However, we have

$$|T_i^- \cup T_j^+| = |T_i^-| + |T_j| - |T_j^-| \geq \begin{cases} l + |T_j| - (l+1) = |T_j| - 1 = t+1 & \text{if } j \neq 1 \\ l + |T_1| - l = |T_1| = t+1 & \text{if } j = 1 \end{cases}$$

and similarly $|T_j^- \cup T_i^+| \geq t+1$, a contradiction.

Thus, even after removing t points, the convex hulls of the sets in $\mathcal{T}$ intersect pairwise. Helly's theorem [6, Theorem 1.3.2] now guarantees that the convex hulls of all sets in $\mathcal{T}$ have a common intersection point. Hence, $\mathcal{T}$ is t-tolerated. □

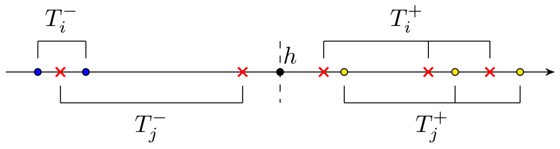


Fig. 3. The convex hulls of two elements in $\mathcal{T}$ are separated after the removal of R. Crosses mark the removed points (i.e., points in R).

Lemma 2.3 immediately gives a way to compute a t-tolerated Tverberg m-partition in $\mathcal{O}(mt \log mt)$ time for $|P| = m(t+2)-1$ by sorting P. However, it is not necessary to know the order of all of P. Algorithm 1 exploits this fact to improve the running time. It repeatedly partitions the point set until it has selected all points whose ranks are multiples of m. These points form the set T_1. Initially, the set Q contains only the input P (line 4). In lines 6–11, we select from each set in Q an element whose rank is a multiple of m (line 8) and we split the set at this element. Here, `select`(P, k) is a procedure that returns the element with rank k of P. After termination of both loops in lines 5–11, all remaining sets in Q correspond to points in P between two consecutive points in T_1. In lines 12–14, the points in the sets in Q are distributed equally among the elements T_i $(i \geq 2)$ of the returned partition.

Algorithm 1. 1d-Tolerated-Tverberg

```
   input : $P \subset \mathbb{R}$, size of partition $m$
1  $r \leftarrow m$;
2  while $r \leq |P|/2$ do
3      $r \leftarrow 2 \cdot r$;
4  $Q \leftarrow \{P\}$; $T_1, T_2, \ldots, T_m \leftarrow \emptyset, \emptyset, \ldots, \emptyset$;
5  while $r \geq m$ do
6      foreach $P' \in Q$ with $|P'| \geq r$ do
7          remove $P'$ from $Q$;
8          $p_r \leftarrow \texttt{select}(P', r)$;
9          $Q \leftarrow Q \cup \{\{p' \in P' \mid p' < p_r\}, \{p' \in P' \mid p' > p_r\}\}$;
10         $T_1 \leftarrow T_1 \cup \{p_r\}$;
11         $r \leftarrow r/2$;
12 foreach $P' \in Q$ do
13     foreach $j \in \{2, 3, \ldots, m\}$ do
14         remove any point from $P'$ and add it to $T_j$;
15 return $\{T_1, T_2, \ldots, T_m\}$;
```

Theorem 2.4. *Let $P \subset \mathbb{R}$ be a set of size $m(t+2)-1$. On input (P, m), Algorithm 1 returns a t-tolerated Tverberg partition for P in time $\mathcal{O}(mt \log t)$.*

Proof. After each iteration of the outer while-loop (lines 5–11), each element $P' \in Q$ has size strictly less than r: initially, Q contains only P and r is strictly greater than $|P|/2$. Hence, both new sets added to Q in line 9 are of size strictly less than r. Since r is halved in each iteration, the invariant is maintained.

We will now check that Lemma 2.3 applies. We only split the sets in Q at elements whose rank is a multiple of m, so the ranks do not change modulo m. By the invariant, after the termination of the outer while-loop in lines 5–11, each set in Q has size strictly less than m. Since the ranks modulo m have not changed, these sets do not contain any element of P whose rank is a multiple of m. Thus, T_1 contains all these elements and the remaining sets in Q after the termination of the outer while-loop in lines 5–11 contain exactly the points of P between two consecutive points of T_1. Lines 12–14, distribute the remaining points among $T_2, \ldots, T_m$. Lemma 2.3 now shows the correctness of the algorithm.

Let us consider the running time. Finding the initial r requires $\mathcal{O}(\log(|P|/m)) = \mathcal{O}(t)$ time. The split-element in line 8 can be found in time $\mathcal{O}(|P'|)$ [3]. Thus, since the sets are disjoint, one iteration of the outer while-loop requires $\mathcal{O}(|P|)$ time, for a total of $\mathcal{O}(\log(|P|/m)|P|) = \mathcal{O}(\log(t)mt)$. By the same argument, both for-loops in lines 12–14 require linear time in the size of P. This results in a total time complexity of $\mathcal{O}(mt \log t)$ as claimed. □

2.2 Higher Dimensions

We use a lifting argument [9] to extend Algorithm 1 to higher-dimensional input. Given a point set $P \subseteq \mathbb{R}^d$ of size n, let h be a hyperplane that splits P evenly (if n is odd, h contains exactly one point of P). We then partition P into $\lfloor n/2 \rfloor$ pairs (p_i^-, p_i^+), where $p_i^- \in h^-$ and $p_i^+ \in h^+$. We obtain a $(d-1)$-dimensional point set with $\lfloor n/2 \rfloor$ elements by mapping each pair to the intersection of the connecting line segment and h.

Let $q_i = p_i^+ p_i^- \cap h$ be the mapped point for (p_i^-, p_i^+) and $\mathcal{T}' = \{T_1', \ldots, T_m'\}$ a t-tolerated Tverberg m-partition of $Q = \{q_1, \ldots, q_{\lfloor n/2 \rfloor}\}$. We obtain a Tverberg m-partition $\mathcal{T}$ with tolerance t for P by replacing each q_i in $\mathcal{T}'$ by its corresponding pair (p_i^-, p_i^+). Thus, we can repeatedly project the set P until Algorithm 1 is applicable. Then, we lift the one-dimensional solution back to higher dimensions.

Algorithm 2 follows this approach. For $d = 1$, Algorithm 1 is applied (lines 1–2). Otherwise, we take an appropriate hyperplane orthogonal to the x_d-axis and compute the lower-dimensional point set (lines 3–7). Finally, the result for $d-1$ dimensions is lifted back to d dimensions (lines 10–11).

Algorithm 2. DimReduct-Tolerated-Tverberg

```
   input  : point set $P \subset \mathbb{R}^d$, tolerance parameter $t$, size of partition $m$
   output: $t$-tolerated Tverberg partition for P of size $m$
 1 if $d = 1$ then
 2     return 1d-Tolerated-Tverberg($P$,$m$)
 3 $h \leftarrow$ hyperplane that halves $P$ according to the $x_d$-coordinate;
 4 foreach $i \in \{1, 2, \ldots, |P \cap h^-|\}$ do
 5     $p_i^- \leftarrow$ remove any point from $P$ that belongs to $P \cap h^-$;
 6     $p_i^+ \leftarrow$ remove any point from $P$ that belongs to $P \cap h^+$;
 7     $q_i \leftarrow$ first $d-1$ coordinates of $p_i^- p_i^+ \cap h$;
 8 $Q \leftarrow \{q_1, q_2, \ldots, q_{|P \cap h^-|}\}$;
 9 $\{T_1', T_2', \ldots, T_m'\} \leftarrow$ DimReduct-Tolerated-Tverberg($Q$,$t$,$m$);
10 foreach $j \in \{1, 2, \ldots, m\}$ do
11     $T_j \leftarrow \{p_i^-, p_i^+ \mid q_i \in T_j'\}$;
12 return $\{T_1, T_2, \ldots, T_m\}$;
```

Proposition 2.5. *Given a set $P \subset \mathbb{R}^d$ of size $2^{d-1}(m(t+2)-1)$, Algorithm 2 computes a t-tolerated Tverberg m-partition for P in time $\mathcal{O}(2^{d-1}dmt + mt \log t)$.*

Proof. Since the size of P halves in each recursion step, 2^{d-1} points suffice to ensure that Algorithm 1 can be applied in the base case. Each projection and lifting step can be performed in linear time, using a median computation. Since the size of the point set decreases geometrically, the total time for projection and lifting is thus $\mathcal{O}(2^{d-1}dmt)$. Since Algorithm 1 has running time $O(mt\log t)$, the result follows. □

For $d \geq 3$, the bound from Proposition 2.5 is worse than the Soberón-Strausz bound. However, in two dimensions, we have

$$2^{2-1}(m(t+2)-1) < (2+1)(m-1)(t+1)+1 \Leftrightarrow m/(m-3) < t$$

This holds for instance if $t \geq 5$ or $m \geq 7$ and $t \geq 2$. Thus, Algorithm 2 gives a strict improvement over the Soberón-Strausz bound for large enough m and t.

3 Reduction to the Untolerated Tverberg Problem

We now show how to use any algorithm that computes (untolerated) approximate Tverberg partitions in order to find tolerated Tverberg partitions. For this, we must increase the tolerance of a Tverberg partition. In the following, we show that one can merge elements of several Tverberg partitions for disjoint subsets of P to obtain a Tverberg partition with higher tolerance for the whole set P. The following lemma is also implicit in the Ph.D. thesis of Colín [4].

Lemma 3.1. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_k$ be Tverberg m-partitions for disjoint point sets P_1, $\ldots$,$P_k \subset \mathbb{R}^d$. Let $T_{i,j}$ be the jth element of $\mathcal{T}_i$ and $t_i \geq 0$ the tolerance of $\mathcal{T}_i$. Then $\mathcal{T} = \{T_j = \bigcup_{i=1}^k T_{i,j} \mid j \in \{1, 2, \ldots, m\}\}$ is a Tverberg m-partition of $P = \bigcup_{j=1}^k P_i$ with tolerance $t = \sum_{i=1}^k t_i + k - 1$.*

Proof. Take $R \subseteq P$ with $|R| = t$. As $t = \sum_{i=1}^k t_i + k - 1 < \sum_{i=1}^k (t_i + 1)$, there is an i with $|P_i \cap R| \leq t_i$. Since $\mathcal{T}_i$ is t_i-tolerated, we have $\bigcap_{i=j}^m \mathsf{conv}(T_{i,j} \setminus R) \neq \emptyset$. Because each $T_{i,j}$ is contained in the corresponding T_j of $\mathcal{T}$, the convex hulls of the elements in $\mathcal{T}$ still intersect after the removal of R. □

This directly implies a simple algorithm: we compute untolerated Tverberg partitions for disjoint subsets of P and then merge them using Lemma 3.1.

Corollary 3.2. *Let $P \subseteq \mathbb{R}^d$ and let $\mathcal{A}$ be an algorithm that computes an untolerated Tverberg m-partition for any point set of size $n_{\mathcal{A}}(m)$ in time $T_{\mathcal{A}}(m)$. Then, a $(\lfloor |P|/n_{\mathcal{A}}(m) \rfloor - 1)$-tolerated Tverberg m-partition for P can be computed in time $\mathcal{O}\left(T_{\mathcal{A}}(n_{\mathcal{A}}(m)) \cdot |P|/n_{\mathcal{A}}(m)\right)$.*

Proof. We split P into $\lfloor |P|/n_{\mathcal{A}} \rfloor$ disjoint sets and use $\mathcal{A}$ to obtain for each subset an untolerated Tverberg partition. Applying Lemma 3.1, we obtain a $(\lfloor |P|/n_{\mathcal{A}} \rfloor - 1)$-tolerated Tverberg m-partition. Since the merging step in Lemma 3.1 takes linear time in $|P|$, the total running time is $\mathcal{O}\left(T_{\mathcal{A}}(n_{\mathcal{A}}(m)) \cdot |P|/n_{\mathcal{A}}(m)\right)$ as claimed. □

Table 1. Corollary 3.2 combined with existing approximation algorithms for the untolerated Tverberg problem

Algorithm	**Tolerance**	**Running time**
Corollary 3.2 with Miller-Sheehy	$\lfloor \lvert P\rvert/2m(d+1)^2\rfloor - 1$	$m^{\mathcal{O}(\log d)}d^{\mathcal{O}(\log d)}\lvert P\rvert$
Corollary 3.2 with Mulzer-Werner	$\lfloor \lvert P\rvert/4m(d+1)^3\rfloor - 1$	$d^{\mathcal{O}(\log d)}\lvert P\rvert$

Table 3 shows specific values for Corollary 3.2 combined with Miller & Sheehy's and Mulzer & Werner's algorithm.

Remark 3.3. Lemma 3.1 gives a quick proof of a slightly weaker version of the Soberón-Strausz bound: partition P into $t+1$ disjoint sets of size at least $\lfloor |P|/(t+1)\rfloor$. By Tverberg's theorem, for each subset there exists an untolerated Tverberg partition of size $\lceil\lfloor |P|/(t+1)\rfloor/(d+1)\rceil$. Using Lemma 3.1, we obtain a t-tolerated Tverberg partition of size $\lceil\lfloor |P|/(t+1)\rfloor/(d+1)\rceil \geq \lceil |P|/(t+1)(d+1)\rceil - 1$ of P, which is at most one less than the Soberón-Strausz bound. This weaker bound was also stated by Colín [4, Lemma 3.3.13].

4 Hardness of Tolerance Testing

Teng [13, Theorem 8.4] proved that testing whether a given point is a centerpoint of a given set (TestingCenter) is coNP-complete if the dimension is part of the input. We show the same for the problem of deciding whether a given Tverberg m-partition has tolerance t (TestingToleratedTverberg) by a reduction to TestingCenter. Here, m can be constant.

Lemma 4.1. *Let $P \subset \mathbb{R}^d$ and let $c \in \mathbb{R}^d$. Then c has depth $t+1$ w.r.t. P if and only if for all subsets $R \subset P, |R| \leq t : c \in \mathsf{conv}(P \setminus R)$.*

Proof. "$\Rightarrow$" Suppose there is some $R \subset P, |R| \leq t$ with $c \notin \mathsf{conv}(P \setminus R)$. Then, there is a half-space h^+ that contains c but no points from $\mathsf{conv}(P \setminus R)$. Thus, $c \in h^+$ and $|P \cap h^+| \leq |R| \leq t$, and hence c has depth $\leq t$ w.r.t. P.

"$\Leftarrow$" Assume c has depth $t' \leq t$ w.r.t. P. Let h^+ be a half-space that contains c and t' points from P. Set $R = h^+ \cap P$. Then, $|R| \leq t$ and $c \notin \mathsf{conv}(P \setminus R)$. □

Theorem 4.2. TestingToleratedTverberg *is coNP-complete if the dimension d and the claimed tolerance t are part of the input.*

Proof. Since testing whether a given partition is Tverberg is a simple application of linear programming, the problem lies in coNP.

Let $(P \subset \mathbb{R}^d, c \in \mathbb{R}^d)$ be an input to TestingCenter. We embed the vector space $\mathbb{R}^d$ in $\mathbb{R}^{d+1}$ by identifying it with the hyperplane $h : x_{d+1} = 0$. Let ℓ

be the line that is orthogonal to h and passes through c. Furthermore, let T^- and T^+ be sets of $t+1$ arbitrary points in $\ell \cap h^-$ and $\ell \cap h^+$, respectively. Set $T = T^- \cup T^+$. We claim that $\{P, T\}$ is a Tverberg 2-partition for $P \cup T$ with tolerance $t = \lceil |P|/(d+1) \rceil - 1$ if and only if c is a centerpoint of P. See Figure 4.

"$\Rightarrow$" Assume $\{P, T\}$ is a t-tolerated Tverberg 2-partition. By construction of T, we have $\mathsf{conv}(P) \cap \mathsf{conv}(T) = \{c\}$. Thus, c lies in the intersection of both convex hulls even if any subset of size at most t is removed. Lemma 4.1 implies that c has depth $t+1 = \lceil |P|/(d+1) \rceil$ w.r.t. P, so c is a centerpoint for P.

"$\Leftarrow$" Assume c is a centerpoint for P. By definition, c has depth at least $\lceil |P|/(d+1) \rceil = t+1$ w.r.t. P. Lemma 4.1 then implies that c is contained in the convex hull of P even if any t points from P are removed. Since T contains $t+1$ points on both sides of a line through c, c is also contained in $\mathsf{conv}(T)$ if any t points from T are removed. Thus, $\{P, T\}$ is a t-tolerated Tverberg 2-partition for $P \cup T$. □

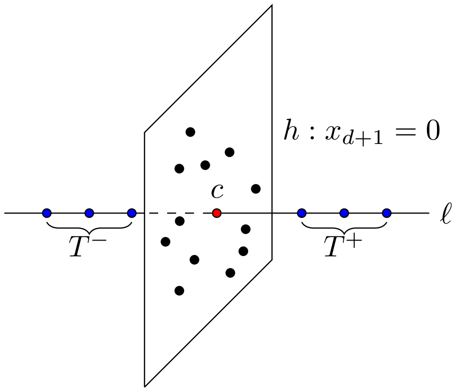


Fig. 4. Reduction of TestingCenter to TestingToleratedTverberg

5 Conclusion

We have shown that each set $P \subset \mathbb{R}$ of size $m(t+2)-1$ can be partitioned into a t-tolerated Tverberg partition of size m in time $\mathcal{O}(mt \log t)$. This is tight, and it improves the Soberón-Strausz bound in one dimension. Combining this with a lifting method, we could also get improved bounds in two dimensions and an efficient algorithm for tolerated Tverberg partitions in any fixed dimension. However, the running time is exponential in the dimension.

This motivated us to look for a way of reusing the existing technology for the untolerated Tverberg problem. We have presented a reduction to the untolerated Tverberg problem that enables us to reuse the approximation algorithms by Miller & Sheehy and Mulzer & Werner.

Finally, we proved that testing whether a given Tverberg partition is of some tolerance t is coNP-complete. Unfortunately, this does not imply anything about the complexity of finding tolerated Tverberg partitions. It is not even clear whether computing tolerated Tverberg partitions is harder than computing untolerated Tverberg partitions. However, we have shown that given a set $P \subset \mathbb{R}^d$ whose size meets the Soberón-Strausz bound, we can obtain in polynomial time a tolerated Tverberg partition from the untolerated Tverberg partition guaranteed by Tverberg's Theorem of size just one less than stated by the Soberón-Strausz bound.

It remains open whether the bound by Soberón and Strausz is tight for $d > 2$. We believe that our results in one and two dimensions indicate that the bound can be improved also in general dimension. Another open problem is finding a *pruning strategy* for tolerated Tverberg partitions. By this, we mean an algorithm that efficiently reduces the sizes of the sets in a t-tolerated Tverberg partition without deteriorating the tolerance. Such an algorithm could be used to improve the quality of our algorithms. In Miller & Sheehy's and Mulzer & Werner's algorithms, Carathéodory's theorem was used for this task. Unfortunately, this result does not preserve the tolerance of the pruned partitions. Also the generalized tolerated Carathéodory theorem [8] does not seem to help. It remains an interesting problem to develop criteria for superfluous points in tolerated Tverberg partitions.

Acknowledgments. We would like to thank the anonymous reviewers for their helpful and detailed comments and for pointing out that the algorithm in Corollary 3.2 could be greatly simplified.

References

[1] Chan, T.M.: An optimal randomized algorithm for maximum Tukey depth. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 430–436 (2004)

[2] Clarkson, K.L., Eppstein, D., Miller, G.L., Sturtivant, C., Hua Teng, S.: Approximating center points with iterative Radon points. International Journal of Computational Geometry & Applications 6, 357–377 (1996)

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)

[4] García Colín, N.: Applying Tverberg Type Theorems to Geometric Problems. PhD thesis, University College London (2007)

[5] Larman, D.: On sets projectively equivalent to the vertices of a convex polytope. Bulletin of the London Mathematical Society 4(1), 6–12 (1972)

[6] Matoušek, J.: Lectures on Discrete Geometry, 1st edn. Springer (2002)

[7] Miller, G.L., Sheehy, D.R.: Approximate centerpoints with proofs. Computational Geometry 43, 647–654 (2010)

[8] Montejano, L., Oliveros, D.: Tolerance in Helly-type theorems. Discrete & Computational Geometry 45, 348–357 (2011)

[9] Mulzer, W., Werner, D.: Approximating Tverberg points in linear time for any fixed dimension. In: Proceedings of the 28th Annual Symposium on Computational Geometry, pp. 303–310 (2012)

[10] Rado, R.: A theorem on general measure. Journal of the London Mathematical Society 1, 291–300 (1946)
[11] Sarkaria, K.: Tverberg's theorem via number fields. Israel Journal of Mathematics 79, 317–320 (1992)
[12] Soberón, P., Strausz, R.: A generalisation of Tverberg's theorem. Discrete & Computational Geometry 47, 455–460 (2012)
[13] Teng, S.-H.: Points, spheres, and separators: a unified geometric approach to graph partitioning. PhD thesis, Carnegie Mellon University Pittsburgh (1992)
[14] Tverberg, H.: A generalization of Radon's theorem. Journal of the London Mathematical Society 41, 123–128 (1966)

Abstract Voronoi Diagrams with Disconnected Regions[⋆]

Cecilia Bohler and Rolf Klein

University of Bonn, Institute of Computer Science I, D-53113 Bonn, Germany

Abstract. Abstract Voronoi diagrams [15, 16] are based on bisecting curves enjoying simple combinatorial properties, rather than on the geometric notions of sites and distance. They serve as a unifying concept. Once the bisector system of any concrete type of Voronoi diagram is shown to fulfill the AVD axioms, structural results and efficient algorithms become available without further effort; for example, the first optimal algorithms for constructing nearest Voronoi diagrams of disjoint convex objects, or of line segments under the Hausdorff metric, have been obtained this way [18]. One of these axioms stated that all Voronoi regions must be pathwise connected, a property quite useful in divide&conquer and randomized incremental construction algorithms. Yet, there are concrete Voronoi diagrams where this axiom fails to hold.

In this paper we consider, for the first time, abstract Voronoi diagrams with disconnected regions. By combining the randomized incremental construction technique [18] with trapezoidal decomposition [21] we obtain an algorithm that runs in expected time $O(s^2 n \sum_{j=3}^{n} m_j/j)$, where s is the maximum number of faces a Voronoi region in a subdiagram of three sites can have, and m_j denotes the average number of faces per region in any subdiagram of j sites. In the connected case, where $s = 1 = m_j$, this results in the known optimal bound $O(n \sum_{j=3}^{n} 1/j) = O(n \log n)$.

Keywords: Abstract Voronoi diagrams, computational geometry, distance problems, trapezoidal decomposition, Voronoi diagrams.

1 Introduction

Voronoi diagrams are useful structures, known in many areas of science. A nice way to think of *concrete Voronoi diagrams* is by expanding circles. Suppose that n local rulers p_i send out their troups, to conquer the plane. Each point z of the plane will belong to that p_i whose troups arrive at z first. Points first reached by two armies simultaneously form Voronoi edges, by three or more, Voronoi vertices. Alterations of the nature of the p_i, the shapes of the circles, individual speeds and starting times, lead to interesting variations of the resulting partitions of the plane; see the surveys and monographs [5–7, 9, 13, 20].

[⋆] This work was supported by the European Science Foundation (ESF) in the EUROCORES collaborative research project EuroGIGA/VORONOI.

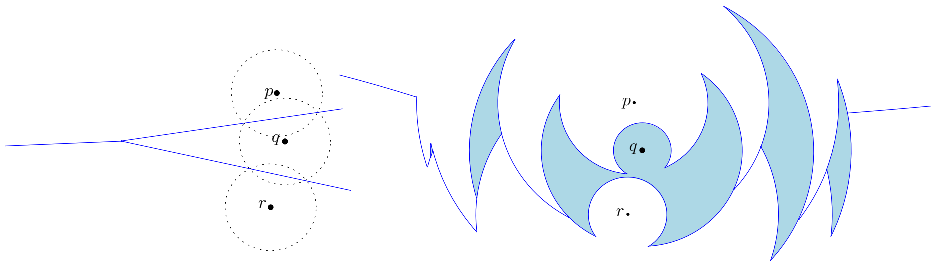


Fig. 1. Two different Voronoi diagrams of three sites. To the right, the different faces of the Voronoi region of q are shaded.

If troops advance in circular formation at the same speed, starting at the same time, the classic Voronoi diagram of n points p_i with respect to the Euclidean distance is obtained; see the left drawing in Figure 1. In the right drawing, circles are expanding at piecewise constant, individual speeds. We observe that the Voronoi region of q consists of several connected components (faces).

Abstract Voronoi diagrams (AVDs, for short) were introduced in Klein [15]. Here, no sites, circles, or distance measures are given. Instead, one takes unbounded curves $J(p, q) = J(q, p)$ as primary objects, together with the open domains $D(p, q)$ and $D(q, p)$ they separate. Abstract Voronoi regions are defined by

$$\mathrm{VR}(p, S) := \bigcap_{q \in S \setminus \{p\}} D(p, q)$$

and the following axioms were required to hold for each subset S' of S.

(A1) Each curve $J(p, q)$, where $p \neq q$, is unbounded. After stereographic projection to the sphere, it can be completed to a closed Jordan curve through the north pole.
(A2) For any two curves $J(p, q)$ and $J(r, t)$, their intersection has only finitely many connected components.
(A3) Each nearest Voronoi region $VR(p, S')$ is pathwise connected.
(A4) Each point of the plane belongs to the closure of a Voronoi region $VR(p, S')$.

It has been shown that the resulting abstract Voronoi diagram $\mathrm{V}(S)$—the plane minus all Voronoi regions— is a planar graph of complexity $O(n)$. It can be constructed, by randomized incremental construction, in $O(n \log n)$ many steps [16–18]. Moreover, the above properties need only be checked for all subsets S' of size three [16]. This makes it easier to verify that a concrete Voronoi diagram falls under the heading of the AVD concept. Examples of such applications can be found in [1,3,4,8,14,18]. Farthest abstract Voronoi diagrams have been studied in [19], and, recently, general order-k AVDs in [10].

In this paper we consider, for the first time, *abstract Voronoi diagrams with disconnected regions.* Instead of Axiom A3 we will only require

(A3') For any $p, q, r \in S$, each Voronoi region in $V(\{p, q, r\})$ has at most s path-connected components.

Axiom A3' implies that two bisecting curves $J(p, q)$ and $J(q, r)$ sharing a site q can cross at most $O(s)$ times, whereas arbitrary bisectors $J(p, q)$ and $J(r, t)$ not sharing a site can have any finite number of intersections. While this axiom bounds the possible number of faces per region in all AVDs of three sites of S, in an AVD of $n > 3$ sites a single region may have as many as $\Theta(sn^2)$ many faces; see Lemma 6 below. To capture the complexity of the input system of bisecting curves we denote by m_j the average number of faces per region, over all AVDs of j sites from S.

Furthermore, to make use of trapezoidal decompositions [21] we will in Section 4 require one additional assumption to bound the number of trapezoids in the decomposition,

(A5) Each bisector $J(p, q)$ has constantly many points of vertical tangency.

This holds e. g. for algebraic functions of constant algebraic degree. Moreover for simplicity we assume general position in the sense that no two distinct points of vertical tangency lie on the same vertical line. This is an assumption commonly made in the design of algorithms for computing arrangements of curves, [22].

In this paper we are proving the following result.

Theorem 1. *Given a system of bisecting curves that satisfies Axioms A1, A2, A3', A4 and A5. Then its abstract Voronoi diagram $V(S)$ can be constructed in expected time*

$$O\left(s^2 n \sum_{j=2}^{n} \frac{m_j}{j}\right).$$

Theorem 1 can be seen as a generalization of the optimal bound obtained in [18] for AVDs with connected regions, because in this case $s = 1 = m_j$ hold and $\sum_{j=3}^{n} 1/j \in O(\log n)$. However, it seems that the standard randomized incremental construction method does not generalize to the disconnected case in a straightforward way. One difficulty is that more conflict information must be stored, in order to locate all faces of the region of a new site. But then one cannot easily bound the outdegree of the history graph. We overcome this difficulty by maintaining a trapezoidal decomposition [21] of the current AVD during the incremental construction.

The rest of this paper is organized as follows. In Section 2 we present some basic facts and complexity results about AVD's with disconnected regions. In Section 3 we discuss preliminary observations with respect to an algorithm. We proceed to studying it in Section 4 and carry out the analysis in Section 5.

2 Preliminaries

In this section we first present some basic facts about abstract Voronoi diagrams with disconnected regions which are akin to facts about the old AVD's with connected regions and can be proved similarly, see Lemma 5 in [16] resp. Lemma 2.2.4 in [15]. The second part of this section discusses some complexity results of the Voronoi diagram. Again let our given system of bisecting curves satisfy axioms A1, A2, A3' and A4 and let s denote the maximum number of connected components of a Voronoi region $\mathrm{VR}(p, S')$ where $p \in S' \subseteq S$, $|S'| = 3$, $|S| = n$.

Lemma 1. *Let $p, q, r \in S$. Then $D(p, q) \cap D(q, r) \subseteq D(p, r)$.*

Lemma 2. *The pathwise connected components of a Voronoi region and its closure are simply connected.*

Theorem 2. *$V(S)$ is a finite planar graph with $O(\#$faces of $V(S))$ vertices and edges.*

Because $V(S)$ is a finite planar graph, the following "piece of pie" lemma can be shown which compares with Lemma 11 in [16].

Lemma 3. *For each point v in the plane there exists an arbitrarily sthemall neighborhood U of v, whose boundary is a simple closed curve, such that the following holds for each subset S' of S. Let $v \in V(S')$. If v is interior point of some Voronoi edge e of $V(S')$ separating the Voronoi regions of p and q then $p \neq q$ and U is divided by e in exactly two domains, one contained in $VR(p, S)$, the other in $VR(q, S)$. Otherwise v is a Voronoi vertex of $V(S')$, of degree ≥ 3. After suitably renumbering S', the Voronoi edges e_i adjacent to v separates the Voronoi regions of p_i and p_{i+1} in clockwise order, where $0 \leq i \leq k-1$ is counted mod k. The edges e_{i-1} and e_i together with ∂U, bound a piece of pie contained in $VR(p_i, S')$; these pieces are domains with Jordan curve boundaries. The sites $p_0, \ldots, p_{k-1}$ do not have to be pairwise different but $p_0 p_1 \ldots p_{k-1}$ is a Davenport-Schinzel sequence of order 2 and if $p_i = p_j$ for $i \neq j$, then the pieces of pie of p_i and p_j belong to different faces of the Voronoi region of $p_i = p_j$.*

The difference from AVDs with connected regions is that several faces of the same region may be incident to the same vertex, as depicted in Figure 2c.

Lemma 4. *$V(\{p, q, r\})$ has at most $6s - 4$ Voronoi vertices, i. e. $J(p, q)$ and $J(p, r)$ can intersect in at most $6s - 4$ points that result in a Voronoi vertex in $V(\{p, q, r\})$. Further this bound is tight.*

Let x be a connected component of the intersection of two bisectors $J(p, q) \cap J(r, t)$. We say that $J(p, q)$ and $J(r, t)$ intersect *transversally* in x, if $J(r, t) \subset D(p, q)$ right before x and $\subset D(q, p)$ right after x or vice versa. Because of the previous Lemma we can in the following assume that each pair of bisectors $J(p, q)$ and $J(p, r)$ intersect transversally at most $6s - 4$ times and each transversal intersection results in a Voronoi vertex of $\mathrm{V}(\{p, q, r\})$.

In the next lemma we use the notation $\lambda_s(n)$ which stands for the maximum length of a (n, s) *Davenport-Schinzel sequence*, see [22]. The idea of how to prove the lower bound in the following lemma is from Agarwal [2].

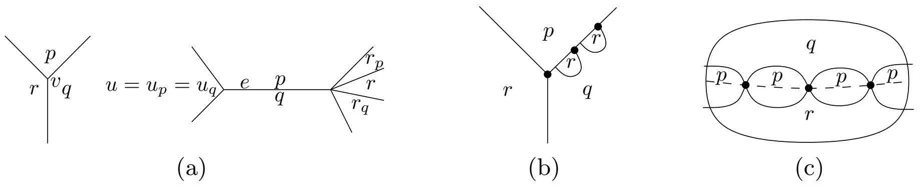


Fig. 2. (a) A pqr-vertex v and $prqt$-edge e with description $D(e) = \{(r_q, q, p, r_p), (u_p, p, q, u_q)\}$. (b) There can be s different pqr-vertices in $\mathrm{V}(\{p, q, r\})$. (c) Touch points with respect to t on an edge of $\mathrm{V}(R \setminus \{t\})$.

Lemma 5. *Let P be a face of $V(S)$. Then P has a complexity of $O(\lambda_{6s-4}(n-1))$. For $s \geq 4$, there exists an AVD having a face of complexity $\Omega(\lambda_{s-1}(n-1))$.*

We observe that this lemma does not imply the same complexity bound for a whole region of a site p, quite in contrary a region with several faces can have complexity $\Theta(sn^2)$.

Lemma 6. *The number of faces and complexity of a region of $V(S)$ is $O(sn^2)$. This bound is tight.*

Lemma 7. *The complexity and number of faces of $V(S)$ is $O(sn^3)$.*

Theorem 3. *Let $J := \{J(p, q) : p \neq q \in S\}$, $|S| = n$, be a curve system fulfilling axioms A1, A2, A3' and A4 for all subsets $S' \subseteq S$ of size 3. Then J fulfills these axioms for all subsets $S' \subseteq S$ of size ≥ 3 and each Voronoi region has $O(sn^2)$ faces.*

3 Towards an Algorithm

We are presenting a randomized incremental algorithm that computes abstract Voronoi diagrams with disconnected regions. Let $R \subset S$, t a site of S not in R, and $\mathcal{T} = \mathrm{VR}(t, R \cup t)$ its new Voronoi region. Allowing Voronoi regions to be disconnected causes some new technical phenomena that did not occur in the connected case [17]. For example, the intersection of a face of the new region $\mathcal{T}$ with the old Voronoi diagram $\mathrm{V}(R)$ need not be connected. Or there may be so-called "touch points" where different faces of the same region meet (Figure 2c).

All these difficulties can be dealt with. Fortunately, it is still true that the fate of an edge e of $\mathrm{V}(R)$ upon insertion of site t can be decided locally. Suppose that in $\mathrm{V}(R)$ edge e separates faces of p and q, and that its endpoints are defined by two more sites r and u. Then e is intersected by the new region of t in $\mathrm{VR}(t, R \cup \{t\})$ if and only if this holds in the Voronoi diagram of the five sites p, q, r, u and t. Therefore, our algorithm will be built on a basic operation which is only slightly more complicated than the one used in [17]. Observe that Voronoi edges e and Voronoi regions $\mathrm{VR}(p, S)$ are defined as open sets i. e. e does not

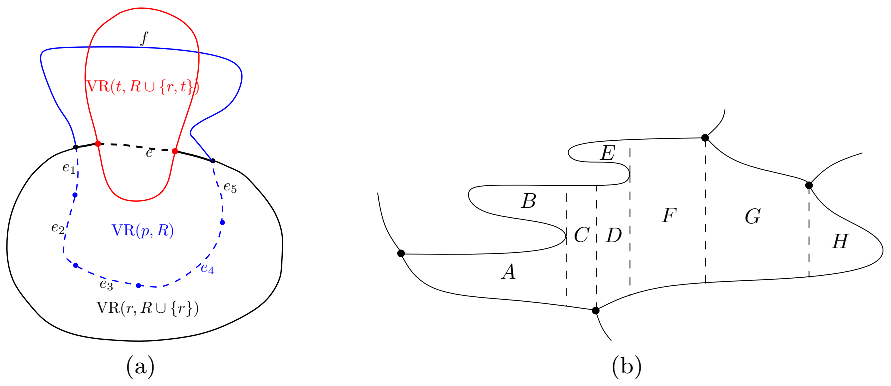


Fig. 3. (a) The sites p, r, t are inserted in this order, the intersection between the region of t and the edge e is not found. (b) Trapezoidal Decomposition of a face.

contain its endpoints and $\mathrm{VR}(p, S)$ does not contain its boundary. With $\overline{\mathrm{VR}(p, S)}$ we denote the closure of the Voronoi region.

Basic Operation

Input: A 5-tuple (p, r, q, u, t), such that

(1) $\mathrm{V}(\{p, r, q, u\})$ contains a $prqu$-edge e, and
(2) $t \notin \{p, r, q, u\}$.

Output: The combinatorial structure of $e \cap \overline{\mathrm{VR}(t, \{p, r, q, u, t\})}$ in the form of:

- Number of connected components of $e \cap \overline{\mathrm{VR}(t, \{p, r, q, u, t\})}$,
- The placement of the connected components of $e \cap \overline{\mathrm{VR}(t, \{p, r, q, u, t\})}$ on e, a special case is $e \subset \overline{\mathrm{VR}(t, \{p, r, q, u, t\})}$.
- Touch points with respect to t on e.

The intersection $e \cap \overline{\mathrm{VR}(t, \{p, r, q, u, t\})}$ can have at most $O(s)$ connected components. Moreover one can show that there can be at most $s - 1$ touch points with respect to t on an edge e of $\mathrm{V}(R)$. That is why we charge $O(s)$ time to each call of the Basic Operation.

However, of much greater concern is that the history graph used in [17] for finding the edges e of $\mathrm{V}(R)$ intersected by the region of the new site, t, does no longer work. In Figure 3a, an example is depicted. Here, the sites p, r, t are inserted in this order. The edge e emerges after inserting r and is made a successor of the edges $e_1, \ldots, e_5$ in the history graph. The region of t intersects edge e but none of the edges $e_1, \ldots, e_5$. This can happen because $\mathcal{T}$ disconnects the region of p, implying that we will not be able to find this intersection by walking through the history graph along the edges intersected by $\mathcal{T}$.

Thus we need a history graph with more information but still with bounded outdegrees. For this purpose we will maintain a trapezoidal decomposition. To make sure that the number of trapezoids is bounded we will in the following require the additional axiom (A5). Furthermore for simplicity we assume that no two points of vertical tangency lie on the same vertical line.

4 Trapezoidal Decomposition

Let $\mathrm{V}^*(R)$ be the vertical decomposition of $\mathrm{V}(R)$, i. e. $\mathrm{V}^*(R)$ decomposes each face of $\mathrm{V}(R)$ into pseudo-trapezoidal cells, for brevity call them trapezoids, see Figure 3b. Such decompositions have been introduced by Seidel [21] and they are also used in [22] to compute arrangements. Again edges e and trapezoids A of $\mathrm{V}^*(R)$ are defined as open sets i. e. e does not contain its endpoints and A does not contain its boundary.

Definition 1. *We say that a site $t \in S \setminus R$* intersects *an edge e of $V(R)$ or a trapezoid A of $V^*(R)$ if $\overline{\mathcal{T}}$ intersects e or $\overline{A}$ in more than finitely many points.*

There can be edges or trapezoids intersected in only finitely many points by $\overline{\mathcal{T}}$ and thus not being intersected by t in the sense of our definition. These edges and trapezoids are taken into account later during the process of inserting $\mathcal{T}$ into $V^*(R)$. The *description* $D_R(A)$ of a trapezoid A of $\mathrm{V}^*(R)$ is defined as the union of the descriptions of the at most two edges (not vertical lines) of $\mathrm{V}(R)$ bounding it, together with the coordinates of the corners of A making the descriptions unique, see Figure 2a. In the following we will not distinguish between a trapezoid A of $\mathrm{V}^*(R)$ and its description.

Definition 2. *We say that a site $t \in S \setminus R$ is in conflict with an edge e of $V(R)$ or a trapezoid A of $V^*(R)$ if there is no edge or trapezoid with the same description in $V(R \cup \{t\})$ resp. $V^*(R \cup \{t\})$.*

Now we define the data structure, called *history graph*, to determine the trapezoids of $\mathrm{V}^*(R)$ intersected by t. Let $R_i = \{p_1, \dots, p_i\}$ be a set of sites inserted in this order and $R = R_j$. The history graph $\mathcal{H}(R)$ is a DAG with a single source and vertices

$$\begin{aligned}\{\text{source}\} \cup \bigcup_{3\le i\le j} \{D_{R_i}(A) | A \text{ is a trapezoid of } \mathrm{V}^*(R_i)\} \\ \cup \bigcup_{4\le i\le j} \{D_{R_i}(e) | e \text{ is an edge on } \partial\mathrm{VR}(p_i, R_i)\}.\end{aligned}$$

Let a vertex of $\mathcal{H}(R)$ be called *trapezoid-vertex* if it refers to a trapezoid and *edge-vertex* if it refers to an edge. Further each vertex is linked to its corresponding trapezoid or edge in $\mathrm{V}^*(R)$ and vice versa. To construct $\mathcal{H}(R)$ we start with the diagram of three sites ∞, p, q and make the trapezoids of $\mathrm{V}^*(\{\infty, p, q\})$ the successors of the source. Let $t \in S \setminus R$ be the next site to be inserted and let e be an edge on the boundary of $\mathcal{T}$. Then e is made a successor of all trapezoids of $\mathrm{V}^*(R)$ intersected by e in their interior. If e does not intersect any trapezoids, i. e. e runs along edges of $\mathrm{V}(R)$, then e is made a successor of those trapezoids A of $\mathrm{V}^*(R)$ intersected by e on their boundary (minus the corners) such that each ε-neighborhood around e intersects the interior of A which compares with Figure 4. Now let A be a trapezoid of $\mathrm{V}^*(R \cup \{t\})$ which has not already been part of

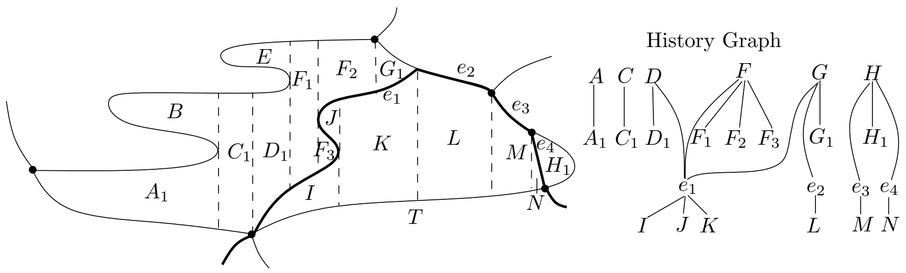


Fig. 4. Face $T \subseteq \mathcal{T}$ is inserted in the trapezoidal decomposition of Figure 3b and the history graph is updated

$\mathrm{V}^*(R)$. If A is contained in a p-region for a $p \in R$, then A is made a successor of all trapezoids A' of $\mathrm{V}^*(R)$ intersected by A. Note that the trapezoids are defined as open sets, hence two trapezoids do not intersect if just their boundaries intersect. Otherwise A is contained in the region of t and is made a successor of the edges on the boundary of $\mathcal{T}$ bordering A. It follows directly that the trapezoids of $\mathrm{V}^*(R)$ are leaves of $\mathcal{H}(R)$ which compares with Figure 4. Let

$$E_t := \{A \text{ trapezoid of } \mathrm{V}^*(R) : A \text{ is intersected by } t\}$$

and

$$\overline{E_t} := \{A \text{ trapezoid of } \mathrm{V}^*(R) : A \text{ is in conflict with } t\}.$$

We have $E_t \subseteq \overline{E_t}$, but there can be trapezoids A contained in $\overline{E_t}$ but not in E_t. This can happen e.g. when an edge bounding A is intersected by t but not A, then there is a trapezoid A' in $\mathrm{V}(R \cup \{t\})$ that has the same coordinates as A but a different description. Note though that $\mathrm{V}^*(R) \cap \overline{\mathcal{T}} \subseteq \bigcup_{A \in E_t} \overline{A} \cap \overline{\mathcal{T}}$. Further let

$$c := \sum_{i=3}^{j} \# \text{ trapezoids of } \mathrm{V}^*(R_i) \text{ in conflict with } t.$$

If an edge-vertex of $\mathcal{H}(R_i)$ is in conflict with t, then none of its trapezoid successors $D_{R_i}(A)$, there is at least one, is part of $\mathrm{V}^*(R_i \cup \{t\})$, i.e. they are all in conflict with t. Thus

$$\begin{aligned} &\# \text{ edge-vertices of } \mathcal{H}(R) \text{ in conflict with } t \\ \leq &\# \text{ trapezoid-vertices of } \mathcal{H}(R) \text{ in conflict with } t \end{aligned}$$

implying

$$\# \text{ vertices of } \mathcal{H}(R) \text{ in conflict with } t \leq 2c.$$

As mentioned before we will first compute the set E_t. Given E_t we will then be able to efficiently insert $\mathcal{T}$ into $\mathrm{V}^*(R)$ and determine the full set $\overline{E_t}$ during the process. To test if a trapezoid $A \subseteq \mathrm{VR}(p, R)$ is intersected by t, we use the Basic Operation to determine if $\overline{\mathcal{T}}$ intersects the edges bounding A in more than finitely many points. To check if the vertical line segments bounding A are intersected by $\overline{\mathcal{T}}$ it is enough to look if the bisector $J(p, t)$ intersects the vertical line segment and if not whether it lies in $D(p,t)$ or $D(t,p)$. Because of our assumption a vertical line test takes constant time. We compute E_t by walking through $\mathcal{H}(R)$ by breadth-first-search along the vertices intersected by t. For all trapezoid-vertices A of $\mathcal{H}(R)$ that appear for the first time in $\mathrm{V}^*(R_i)$, intersected by t, we also test recursively the at most 4 trapezoids of $\mathrm{V}^*(R')$ which are adjacent to A by a vertical line segment for intersection with t.

Lemma 8. *By walking through $\mathcal{H}(R)$ as described above, we will reach all leaves of $\mathcal{H}(R)$ which are intersected by t.*

Lemma 9. *The outdegree of each trapezoid-vertex of $\mathcal{H}(R)$ is $O(s)$.*

Lemma 10. *The set E_t can be computed in time $O(s^2c)$.*

4.1 Construction of $\mathrm{V}^*(R \cup \{t\})$ and $\mathcal{H}(R \cup \{t\})$

At first we discuss the vertices that have to be updated to construct $\mathrm{V}(R \cup \{t\})$ from $\mathrm{V}(R)$. There are new vertices V_{new} that need to be added to $\mathrm{V}(R)$, vertices V_{chang} already existing in $\mathrm{V}(R)$ but getting a new adjacency list in $\mathrm{V}(R \cup \{t\})$ and vertices V_{del} that have to be deleted from $\mathrm{V}(R)$. We can define these vertices as follows

- $V_{unch} := \{$vertices v of $\mathrm{V}(R) : v \notin \overline{\mathcal{T}}\}$
- $V_{chang} := \{$vertices v of $\mathrm{V}(R) : v \in \partial\mathcal{T}$ and not all edges incident to v are clipped at v by t or v is a touch point$\}$
- $V_{del} := \{$vertices v of $\mathrm{V}(R)$: all edges incident to v are clipped at v by t and v is not a touch point$\}$
- $V_{new} := \{$endpoints of $e - \overline{\mathcal{T}}$ or touch points with respect to t on $e\}$.

For all trapezoids $A \subseteq \mathrm{VR}(p,R)$ in E_t we test how the edges and vertical lines bounding A are intersected by $\mathcal{T}$ (using the Basic operation for the edges). To decide whether a vertex v of $\mathrm{V}(R)$ is a touch point with respect to t we have to look at the bisectors $J(t,p_1),\ldots,J(t,p_k)$ for all p_i having a region in $\mathrm{V}(R)$ that contains a trapezoid intersected by t bordering v and test if the region of p_i in an ε-neighborhood around v is intersected by $J(t,p_i)$. Thus we have all the information to directly compute the sets V_{unch}, V_{chang}, V_{del} and V_{new}.

Like in [17] we know how to update the vertices v of $\mathrm{V}(R\cup\{t\})$ along with the incident edges in their clockwise manner around v. In contrary the intersections between the region of t and $\mathrm{V}(R)$ are not connected, not even for one face of $\mathcal{T}$. Hence for each face of $\mathcal{T}$ we traverse its boundary, which is a closed curve, through $\mathrm{V}^*(R)$. This is done by starting at a trapezoid $A \subseteq \mathrm{VR}(p,R)$ intersected by t and following the bisector $J(p,t)$ through A into an adjacent trapezoid $B \subset \mathrm{VR}(q,R)$ and so on until the starting point is reached again.

Lemma 11. *If E_t is known, then $\mathrm{V}^*(R\cup\{t\})$ and $\mathcal{H}(R\cup\{t\})$ can be constructed from $\mathrm{V}^*(R)$ and $\mathcal{H}(R)$ in time $O(s^2|\overline{E_t}|)$.*

5 Analysis

The sets E_t for all $t \in S$ can be computed in time $O(s^2c)$, Lemma 10, and $\mathrm{V}^*(R \cup \{t\})$ and $\mathcal{H}(R \cup \{t\})$ in time $O(s^2|\overline{E_t}|)$, Lemma 11. Thus we need to estimate c and $|\overline{E_t}|$. To shed light on the space consumption the size of $\mathcal{H}(S)$ is required. For a given bisector system let m_j denote the average number of faces per region over all AVDs of j sites from S. We use the analysis for randomized incremental constructions of Clarkson, Mehlhorn, Seidel [12] and apply it to our history graph based on trapezoids.

Lemma 12. *The complexity of $V^*(R)$ lies in $O(|\,V(R)|)$.*

Lemma 13. *For $R = \{r_1, \dots, r_i\}$, the expectation of c is $O(\sum_{j=2}^{n} \frac{m_j}{j})$.*

Lemma 14. *The expected size of $\mathcal{H}(S)$ is $O(\sum_{j=4}^{n} m_j)$.*

Lemma 15. *The expected size of $\overline{E_t}$ is $O(m_j)$.*

Theorem 4. *$V(S)$ can be computed in expected time $O(s^2 n \sum_{j=2}^{n} \frac{m_j}{j})$ and expected space $O(\sum_{j=4}^{n} m_j)$.*

6 Discussion

One question is how m_j affects the performance of our algorithm. There do exist examples where $m_j = j$ and $|\mathrm{V}(S)| \in O(n^2)$, for a constant s, thus the algorithm is output sensitive. If though there is one more site p such that $\mathrm{V}(S) \subset D(p, q)$ for all $q \in S$ then the expected running time is still the same but the size of $\mathrm{V}(S \cup \{p\})$ is constant. For such examples randomization does not help.

Another still open problem is to find a tight upper bound for the size of $\mathrm{V}(S)$. Because of the transitivity, Lemma 1, we can define a total order on S for all $x \in \mathbb{R}^2$ by $p \leq_x q$ iff $x \in D(p, q)$. This leads to the conjecture that the complexity of $\mathrm{V}(S)$ equals the complexity of the lower envelope of surfaces. If each pair of bisectors $J(p, q)$ and $J(r, t)$ intersect in at most s components and s is constant this is $O(n^{2+\varepsilon})$, [22]. But axiom (A3') implies the at most s intersections only for bisectors sharing a site, whereas other bisectors can intersect in any finite number of components.

References

1. Abellanas, M., Hurtado, F., Palop, B.: Transportation Networks and Voronoi Diagrams. In: Proceedings of the International Symposium on Voronoi Diagrams in Science and Engineering (2004)
2. Agarwal, P.: Personal Communication (2012)
3. Ahn, H.-K., Cheong, O., van Oostrum, R.: Casting a Polyhedron with Directional Uncertainty. Computational Geometry: Theory and Applications 26(2), 129–141 (2003)

4. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest Paths, Straight Skeletons, and the City Voronoi Diagram. Discrete and Computational Geometry 31(7), 17–35 (2004)
5. Aurenhammer, F.: Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys 23(3), 345–405 (1991)
6. Aurenhammer, F., Klein, R.: Voronoi Diagrams. In: Sack, J.R., Urrutia, G. (eds.) Handbook on Computational Geometry, pp. 201–290. Elsevier (1999)
7. Aurenhammer, F., Klein, R., Lee, D.-T.: Voronoi Diagrams and Delaunay Triangulations. World Scientific Publishing Company (to appear in August 2013)
8. Bae, S.W., Chwa, K.-Y.: Voronoi Diagrams for a Transportation Network on the Euclidean Plane. International Journal on Computational Geometry and Applications 16, 117–144 (2006)
9. Boissonnat, J.D., Wormser, C., Yvinec, M.: Curved Voronoi Diagrams. In: Boissonnat, J.D., Teillaud, M. (eds.) Effective Computational Geometry for Curves and Surfaces. Mathematics and Visualization. Springer (2006)
10. Bohler, C., Cheilaris, P., Klein, R., Liu, C.-H., Papadopoulou, E., Zavershynskyi, M.: On the Complexity of Higher Order Abstract Voronoi Diagrams. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 208–219. Springer, Heidelberg (2013)
11. Bohler, C., Klein, R.: Point Sites with Individual Distance Functions, Bonn (2012), `http://www.i1.informatik.uni-bonn.de/sites/default/files/ManyDist.pdf` (manuscript)
12. Clarkson, K., Mehlhorn, K., Seidel, R.: Four Results on Randomized Incremental Constructions. Computational Geometry: Theory and Applications 3, 185–212 (1993)
13. Fortune, S.: Voronoi Diagrams and Delaunay Triangulations. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 20, pp. 377–388. CRC Press LLC (1997)
14. Karavelas, M.I., Yvinec, M.: The Voronoi Diagram of Planar Convex Objects. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 337–348. Springer, Heidelberg (2003)
15. Klein, R.: Concrete and Abstract Voronoi Diagrams. LNCS, vol. 400. Springer, Heidelberg (1989)
16. Klein, R., Langetepe, E., Nilforoushan, Z.: Abstract Voronoi Diagrams Revisited. Computational Geometry: Theory and Applications 42(9), 885–902 (2009)
17. Klein, R., Mehlhorn, K., Meiser, S.: Randomized Incremental Construction of Abstract Voronoi Diagrams. Computational Geometry: Theory and Applications 3, 157–184 (1993)
18. Mehlhorn, K., Meiser, S., Ó'Dúnlaing, C.: On the Construction of Abstract Voronoi Diagrams. Discrete and Computational Geometry 6, 211–224 (1991)
19. Mehlhorn, K., Meiser, S., Rasch, R.: Furthest Site Abstract Voronoi Diagrams. International Journal of Computational Geometry and Applications 11(6), 583–616 (2001)
20. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley Series in Probability and Statistics (2000)
21. Seidel, R.: A Simple and Fast Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. Computational Geometry: Theory and Applications 1, 51–64 (1991)
22. Sharir, M., Agarwal, P.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press (1995)

Terrain Visibility with Multiple Viewpoints[⋆]

Ferran Hurtado[1], Maarten Löffler[2], Inês Matos[1,4], Vera Sacristán[1], Maria Saumell[5], Rodrigo I. Silveira[3,1], and Frank Staals[2]

[1] Dept. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain
[2] Dept. of Information and Computing Sciences, Universiteit Utrecht, The Netherlands
[3] Dept. de Matemática, Universidade de Aveiro, Portugal
[4] Dept. de Matemática & CIDMA, Universidade de Aveiro, Portugal
[5] Département d'Informatique, Université Libre de Bruxelles, Belgium

Abstract. We study the problem of visibility in polyhedral terrains in the presence of multiple viewpoints. We consider three fundamental visibility structures: the visibility map, the colored visibility map, and the Voronoi visibility map. We study the complexity of each structure for both 1.5D and 2.5D terrains, and provide efficient algorithms to construct them. Our algorithm for the visibility map in 2.5D terrains improves on the only existing algorithm in this setting.

1 Introduction

Visibility problems or, to be more specific, problems regarding whether two objects are *visible* from each other amidst a number of obstacles have been a hot topic in computational geometry. In this paper we are interested in visibility on *terrains*. A *2.5D terrain* is an xy-monotone polyhedral surface in $\mathbb{R}^3$. We also study 1.5D terrains: x-monotone polygonal lines in $\mathbb{R}^2$. The obstacles we consider are the terrain edges or triangles themselves. A fundamental aspect of visibility in terrains is the *viewshed* of a point (i.e. the *viewpoint*): the (maximal) regions of the terrain that the viewpoint can see.

In a 1.5D terrain, the viewshed is almost equivalent to the *visibility polygon* of a viewpoint, so well-known linear-time algorithms can be applied. In 2.5D the viewshed is more complex (see Fig. 1). In an n-vertex terrain, the viewshed of a viewpoint can have $\Theta(n^2)$ complexity. The best algorithms known to compute it take $O((n+k)\log n \log\log n)$ time [13], and $O((n\alpha(n)+k)\log n)$ time [9], where k is the size of the resulting viewshed, and $\alpha(n)$ is the inverse of the Ackermann function.

While the computation of the viewshed from one viewpoint on a terrain has been thoroughly studied, it is surprising that a natural and important variant has been left open: What happens if instead of *one* single viewpoint, one has *many*, say $m > 1$,

⋆ F. H., V. S., and R.I. S. were partially supported by projects Gen. Cat. DGR 2009SGR1040, MINECO MTM2012-30951 and ESF EuroGIGA-ComPoSe - MICINN EUI-EURC-2011-4306. M. L. and F. S. were supported by Netherlands Organisation for Scientific Research (NWO) under project no 639.021.123 and 612.001.022. I. M. was supported by FCT project PEst-C/MAT/UI4106/2011 with COMPETE number FCOMP-01-0124-FEDER-022690. M. S. was supported by projects F.R.S.-FNRS - EUROGIGA NR 13604 and GAČR GIG/11/E023. R.I. S. was partially supported by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235, and by FCT through grant SFRH/BPD/88455/2012.

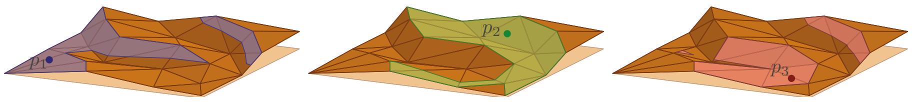


Fig. 1. The viewsheds of three viewpoints on a 2.5D terrain

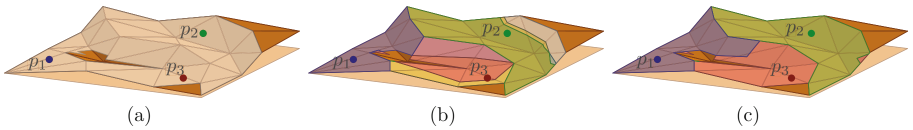


Fig. 2. The visibility map (a), the colored visibility map (b), and the Voronoi visibility map (c)

different viewpoints on the terrain? The *common viewshed*, or *visibility map* can then be defined as the regions of the terrain that can be seen from *at least one* viewpoint. Computing the viewshed from each single viewpoint and then taking the union of the m viewsheds is a straightforward solution, but it has a high running time that does not take the final size of the visibility map into account. Obtaining more efficient algorithms for this and other related problems is the main focus of this paper.

To the best of our knowledge, there are no other studies in computational geometry on the visibility map of multiple viewpoints. We are not aware of any work for 1.5D terrains, whereas for 2.5D terrains we can only mention [6], where they essentially overlay the m individual viewsheds without studying the complexity of the visibility map. This results in the high running time of $O(m^2n^4)$. In addition, a few papers deal with the computation of viewsheds for multiple viewpoints for rasterized terrains [6,11].

We would like to highlight the fact that it is not due to its lack of interest that visibility from multiple viewpoints has been overlooked up to now. Visibility in 1.5D terrains has been thoroughly studied from related perspectives, and in particular the problem of *placing* a minimum number of viewpoints to cover a terrain has received a lot of attention (e.g. [2,3,5,7,10]). Their theoretical interest and the fact that 1.5D terrains already pose a difficult challenge are the main motivation behind our work in that dimension.

Regarding 2.5D, the applications are too numerous to be detailed here, so we only present a few concrete examples. For instance, evaluating the effectiveness of a set of fire lookout towers [4], or identifying locations for placing wind turbines so they are not visible from "sensitive sites" like touristic points [12]. Finally, our results also apply to other contexts like sensor networks, in which wireless devices have to be placed on a terrain, and we have to measure the quality of the chosen device placement scheme [14]. The structures we study are particularly interesting within this context.

Problem Statement. A 2.5D terrain $\mathcal{T}$ consists of a set $V(\mathcal{T})$ of n vertices, a set $E(\mathcal{T})$ of $O(n)$ edges, and a set $F(\mathcal{T})$ of $O(n)$ faces. A 1.5D terrain $\mathcal{T}$ consists of a set $V(\mathcal{T})$ of n vertices and a set $E(\mathcal{T})$ of $n-1$ edges.

For any point p on the terrain $\mathcal{T}$ (either a 2.5D terrain or a 1.5D terrain), the *viewshed* of p on $\mathcal{T}$, denoted by $\mathcal{V}_{\mathcal{T}}(p)$, is the maximal set of points on $\mathcal{T}$ that are *visible* from p. A point q is visible from p if and only if the line segment $\overline{pq}$ does not intersect any point strictly below the terrain surface (intuitively, this corresponds to placing the viewpoints some small $\varepsilon > 0$ above the terrain). Note that our definition of visibility is symmetric, and that viewpoints have unlimited sight. The viewshed $\mathcal{V}_\mathcal{T}(\mathcal{P})$ of a set of viewpoints $\mathcal{P}$ is the set of points visible from at least one viewpoint in $\mathcal{P}$.

Given a set of viewpoints $\mathcal{P}$, we define the *Voronoi viewshed* $\mathcal{W}_\mathcal{T}(p, \mathcal{P})$ of a viewpoint $p \in \mathcal{P}$ as the set of points in the viewshed of p that are closer to p than to any other viewpoint that can see them. More precisely, $\mathcal{W}_\mathcal{T}(p, \mathcal{P}) = \mathcal{V}_\mathcal{T}(p) \cap \{x \mid x \in \mathcal{T} \wedge \mathit{closest}_\mathcal{T}(x, \mathcal{P}) = p\}$, where $\mathit{closest}_\mathcal{T}(x, \mathcal{P})$ denotes the closest (in terms of the Euclidean distance) viewpoint in $\mathcal{P}$ that can see a point x on $\mathcal{T}$.

We study three fundamental terrain visibility structures regarding multiple viewpoints for 1.5D and 2.5D terrains. These structures are illustrated in Fig. 2.

The *visibility map* $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ is a subdivision of the terrain $\mathcal{T}$ into a visible region $R_V = \mathcal{V}_\mathcal{T}(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} \mathcal{V}_\mathcal{T}(p)$ and an invisible region $R_I = \mathcal{T} \setminus R_V$.

The *colored visibility map* $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$ is a subdivision of the terrain $\mathcal{T}$ into maximally connected regions R, each of which is covered by exactly the same subset of viewpoints $\mathcal{P}' \subseteq \mathcal{P}$. Each region R is a (maximally connected) subset of $\bigcap_{p \in \mathcal{P}'} \mathcal{V}_\mathcal{T}(p)$ and we have that $R \cap \bigcup_{p \in \mathcal{P} \setminus \mathcal{P}'} \mathcal{V}_\mathcal{T}(p) = \emptyset$.

The *Voronoi visibility map* $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$ is a subdivision of the terrain $\mathcal{T}$ into maximally connected regions, each of which is a subset of the Voronoi viewshed $\mathcal{W}_\mathcal{T}(p, \mathcal{P})$ of a viewpoint $p \in \mathcal{P}$.

We denote the *size*, that is, the total complexity of all its regions, of $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$, $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$, and $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$, by k, k_c, and k_v, respectively.

For simplicity, we assume that $\mathcal{P}$ is a set of m viewpoints placed on terrain vertices, thus $m \leq n$. We consider this a reasonable assumption, since in most applications the number of terrain vertices is considerably larger than the number of viewpoints.

Results. We present a comprehensive study of the visibility structures defined above. We analyze the complexity of all the structures and propose algorithms to compute them. Our results are summarized in Table 1. Regarding 1.5D terrains, all our algorithms avoid computing individual viewsheds. $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ is computed in nearly optimal running time, while the algorithms for $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$ and $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$ are output-sensitive. Obtaining the latter algorithm, whose running time depends on k_c and k_v, was surprisingly challenging, and required using several subtle geometric properties of the problem.

Table 1. Complexity and computation time of the three visibility structures

	1.5D Terrains		2.5D Terrains	
Structure	Max. size	Computation time	Max. size	Computation time
Vis	$\Theta(n)$	$O(n \log n)$	$O(m^3 n^2)$	$O(m(n\alpha(n) + k_c) \log n)$
ColVis	$\Theta(mn)$	$O(n + (m^2 + k_c) \log n)$	$O(m^3 n^2)$	$O(m(n\alpha(n) + k_c) \log n)$
VorVis	$\Theta(mn)$	$O(n + (m^2 + k_c) \log n + k_v(m + \log n \log m))$	$O(m^4 n^2)$	$O(m(n\alpha(n) + k_c) \log n)$

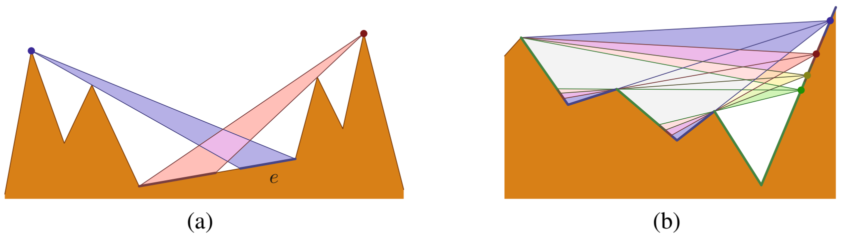


Fig. 3. (a) Edge e contains one invisible connected portion between two visible ones. (b) Every other edge has four different regions of $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$ and four different regions of $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$.

As for 2.5D terrains, we prove with a careful analysis—interesting on its own—that the maximum complexity of $\mathrm{Vis}(\mathcal{T},\mathcal{P})$ and $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$ is much less than the overlay of the viewsheds, as implicitly assumed in previous work [6]. Using that, we show how a combination of well-known algorithms can be used to compute the visibility structures reasonably fast. Omitted proofs and details are given in the full version of this paper [8].

2 1.5D Terrains

2.1 Complexity of the Visibility Structures

In 1.5D our visibility structures can be seen as subdivisions of the x-axis into intervals.

Theorem 1. *Given a 1.5D terrain $\mathcal{T}$:* $\mathrm{Vis}(\mathcal{T},\mathcal{P})$ *has maximum complexity $\Theta(n)$, and* $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$ *and* $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$ *both have maximum complexity $\Theta(mn)$.*

Proof (Sketch). There are two types of points of $\mathcal{T}$ that contribute to the complexity of $\mathrm{Vis}(\mathcal{T},\mathcal{P})$: vertices of $\mathcal{T}$, and points where the $\mathcal{T}$ goes from visible to invisible or vice versa. There are n points of the first type. The points of the second type amount to $O(n)$, since it is easy to see that the interior of every edge $e \in E(\mathcal{T})$ contains at most two such points (see Fig. 3(a) for an example). Consequently, k is $\Theta(n)$.

As for $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$, notice that once a viewpoint sees a given point q on an edge $e \in E(\mathcal{T})$, it must see the whole segment from q to one of its endpoints. Hence, e can be split into at most $m+1$ different regions of $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$. Therefore the complexity of $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$ is $O(mn)$. The example in Fig. 3(b) shows that this is tight.

Finally, let us focus on $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$. For a given edge $e \in E(\mathcal{T})$, $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$ restricted to e has at most $4m-2$ regions (Lemma 1 of [8]), thus implying the upper bound. The lower bound can be achieved by a configuration of viewpoints on a particular terrain $\mathcal{T}$ that can be repeated so that every other edge of $\mathcal{T}$ has as many Voronoi regions as viewpoints, for arbitrary n and m. An example is shown in Fig. 3(b). □

2.2 Algorithms to Construct the Visibility Structures

Construction of the Visibility Map. To construct the visibility map we first compute the left- and right-visibility maps, and then merge them. The *left(right)-visibility map* partitions $\mathcal{T}$ into two regions: the visible and the invisible portions of the terrain, where *visible* means visible from a viewpoint *to the left (right)* of that point of the terrain

In the following we explain the construction of the left-visibility map (thus, *visible* stands for *left-visible*). The algorithm uses the following property of 1.5D terrains, which is a consequence of the so-called *order claim* (See Claim 2.1 in [2]):

Observation 1. *Let $q \in \mathcal{T}$ be a point visible from the left by p_i and p_j, with p_i to the left of p_j. For any $r \in \mathcal{T}$ to the right of q, if p_i does not see r, then p_j cannot see r either.*

The algorithm sweeps the terrain from left to right while maintaining the leftmost visible viewpoint (if any), which we call the *active* viewpoint. The algorithm also stores a priority queue of events that comprises the x-coordinates of the vertices (*vertex events*) and the points of the terrain where a viewpoint becomes visible (*viewpoint events*). Initially we add an event for each terrain vertex and viewpoint, the latter corresponding to the position of the viewpoint on the terrain. We process the events sorted by their x-coordinate. When two events have the same x-coordinate, viewpoint events are processed first. Let p_a be the active viewpoint (if no viewpoint is visible, we set $p_a = \perp$).

(i) Viewpoint event, for a viewpoint p_i. If $p_a = \perp$, then a new visibility region starts. If $p_a = \perp$ or p_i is to the left of p_a, then p_i becomes the active viewpoint.

(ii) Vertex event, for a vertex v. If the active viewpoint p_a becomes invisible after v, we compute where p_a becomes visible again by a ray-shooting query, and add a viewpoint event there. If there was a viewpoint event at v as well, this viewpoint becomes the active viewpoint. Otherwise, the current visibility region ends at v.

The correctness of the method follows from Obs. 1, which guarantees that it is enough to keep track of only the leftmost visible viewpoint. The following theorem is proved in the full version.

Theorem 2. *Given a 1.5D terrain $\mathcal{T}$, the visibility map $\mathrm{Vis}(\mathcal{T}, P)$ can be constructed in $O(n \log n)$ time.*

Construction of the Colored Visibility Map. The computation of the colored visibility map is similar to that of $\mathrm{Vis}(\mathcal{T}, P)$, with the extra complication of having to maintain all visible viewpoints during the sweep. We show in the full version that we can still handle each event in $O(\log n)$ time. In principle, the event processing time can be charged to the output size k_c –since each viewpoint is likely to generate a new region when it reappears. However, it can happen that several viewpoints reappear at exactly the same point, generating a single region in ColVis. With some analysis we show that the total number of these situations is $O(m^2)$, leading to the following result.

Theorem 3. *Given a 1.5D terrain $\mathcal{T}$, the colored visibility map $\mathrm{ColVis}(\mathcal{T}, P)$ can be constructed in $O(n + (m^2 + k_c) \log n)$ time.*

Construction of the Voronoi Visibility Map.

Divide and Conquer Approach. A way to construct $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$ consists in dividing the set of viewpoints into two subsets, computing the Voronoi visibility map of the two subsets recursively, and merging the two maps. This takes $O(mn \log m)$ time.

An Output-Sensitive Algorithm. Even though $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$ can have $\Theta(mn)$ complexity, it seems unlikely that such high complexity arises often in practical applications. In the following we present an alternative algorithm that essentially extracts the Voronoi visibility map from the colored visibility map. Its running time depends on the complexity of the two structures, and avoids the fixed $O(mn)$ term of the previous method.

The algorithm sweeps the terrain from left to right. During this sweep, we maintain three data structures: (i) a doubly-linked list with the vertices of $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$, ordered from left to right, (ii) a list $\mathcal{P}'$ with the currently visible viewpoints, and (iii) for each $p_i \in \mathcal{P}'$, the starting point a_i of the last region in which p_i is visible encountered so far in the sweep. We will use $\mathcal{T}[a,c]$, for a,c on $\mathcal{T}$ and $x(a) < x(c)$, to denote the closed portion of the terrain between a and c. The algorithm produces $\mathrm{VorVis}(\mathcal{T},\mathcal{P})$ as a list of interval, viewpoint pairs $([a,c],p_i)$, such that p_i is the closest viewpoint to all points in $\mathcal{T}[a,c]$. If $\mathcal{T}[a,c]$ is not visible from any viewpoint, p_i is set to $\perp$.

Our algorithm uses the following two functions, whose implementation is described later. $\textsc{IsAlwaysCloser}([a,c],p_1,p_2)$ determines whether p_1 is always closer than p_2 in $\mathcal{T}[a,c]$, assuming both viewpoints are visible throughout $\mathcal{T}[a,c]$. $\textsc{FirstRegionChange}([a,c],p_1,\mathcal{P}')$ assumes that p_1 is visible throughout $\mathcal{T}[a,c]$ and is the closest visible viewpoint at a; it returns the leftmost point in $\mathcal{T}[a,c]$ where p_1 stops being the closest visible viewpoint from $\mathcal{P}'$ (or the end of the interval, if that never happens).

We process $\mathcal{T}$ in a number of iterations. Each iteration starts at the leftmost point u of a new Voronoi region, with $\mathcal{P}'$ containing the viewpoints that are visible from u.

If $\mathcal{P}' = \emptyset$, then the region starting at u and ending at the start point v of the next region in $\mathrm{ColVis}(\mathcal{T},\mathcal{P})$ is not visible from any viewpoint. We report the region $[u,v]$ with $\perp$, and move forward (towards the right) until v, where a new Voronoi region, and thus a new iteration, starts.

If $\mathcal{P}' \neq \emptyset$, we compute the closest visible viewpoint in $O(m)$ time; if there is more than one, we move infinitesimally to the right of u, and compute the closest visible viewpoint there. Without loss of generality, we assume that the closest visible viewpoint is p_1. For all viewpoints $p_i \in \mathcal{P}'$, we set $a_i := u$. We now start traversing the terrain, from u towards the right. At a point q, we might find several events from ColVis:

1. A viewpoint p_j becomes visible. We update $\mathcal{P}'$, set $a_j := q$, and continue the sweep.
2. A viewpoint $p_j \neq p_1$ becomes invisible. We update $\mathcal{P}'$ and proceed depending on two subcases:
   (a) $\textsc{IsAlwaysCloser}([a_j,q],p_1,p_j)$ = $\textsc{True}$. Continue traversing the terrain.
   (b) $\textsc{IsAlwaysCloser}([a_j,q],p_1,p_j)$ = $\textsc{False}$. There is a point in $\mathcal{T}[a_j,q]$ at which p_j is closer than p_1, so at least one Voronoi region starts between u and q. We find the leftmost region change v by calling $\textsc{FirstRegionChange}([u,q], p_1,\mathcal{P}')$, and report $[u,v]$ as a Voronoi region with p_1 as closest point. We now backtrack our sweep, i.e. we traverse the terrain from right to left (updating $\mathcal{P}'$ as we encounter events), until we reach v, and start a new Voronoi region, and thus a new iteration of our algorithm at v.
3. Viewpoint p_1 becomes invisible. We update $\mathcal{P}'$, and compute $\textsc{IsAlwaysCloser}$ $([a_i,q],p_1,p_i)$, for all $p_i \in \mathcal{P}'$. If the answer is $\textsc{True}$ for *all* viewpoints in $\mathcal{P}'$, we report the region $[u,q]$ with p_1 as closest viewpoint, and start a new Voronoi region

and a new iteration at q. Otherwise, there is at least one Voronoi region that starts between u and q. We handle this analogously to case 2(b).

After processing the events of type 2 at the rightmost vertex of the terrain, we have successfully computed $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$. Since we backtrack our sweep in step 2, it may be the case that we (unnecessarily) visit events from $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$ multiple times. We can avoid this, by augmenting this step as follows. Consider step 2a. We notice that there cannot be a Voronoi region of p_j between a_j and q (since at least p_1 is closer and visible). So we can remove the events of p_j becoming visible at a_j and invisible at q from $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$. We remove q in step 2a itself. Event a_j is removed if we encounter it while backtracking in step 2b: at each event of type 1, i.e. a viewpoint p_j becoming visible, we check if p_j is in $\mathcal{P}'$. If not, we must have removed its corresponding endpoint (i.e. q) from $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$. Thus we can also remove a_j.

As for the auxiliary functions, $\textsc{IsAlwaysCloser}([a_j, q], p_1, p_j)$ can be implemented to run in $O(\log n)$ time by doing a ray-shooting query, where the ray is the bisector of p_1 and p_j. However, it is possible to answer this question faster.

Lemma 1. *Consider two points r and t such that all of $\mathcal{T}[r, t]$ is visible from two viewpoints p_1 and p_2. We can decide whether there exists some point in $\mathcal{T}[r, t]$ that is closer to p_2 than to p_1 in $O(1)$ time.*

$\textsc{FirstRegionChange}([u, q], p_1, \mathcal{P}')$ can be implemented to run in $O(m \log n)$ time as follows: For every $p_i \in \mathcal{P}'$, and using a ray-shooting query, compute the leftmost point (if any) on $\mathcal{T}[a_i, q]$ that is closer to p_i than to p_1. Then keep the leftmost point u' among all the points encountered. Again, it is possible to do this faster:

Lemma 2. *Let $[u, q]$ be an interval such that $p_1 \in \mathcal{P}$ is visible in all $\mathcal{T}[u, q]$ and is the closest visible viewpoint at u. Let $\mathcal{P}'$ be a set of viewpoints such that for each $p_i \in \mathcal{P}'$, $\mathcal{T}[a_i, q]$ is visible from p_i, for some a_i such that $x(u) \leq x(a_i)$. Then in $O(m + \log n \log m + n')$ time we can find the leftmost point $u' \in \mathcal{T}[u, q]$ such that at u' there is a change of region in $\mathrm{VorVis}(\mathcal{T}, \mathcal{P}')$, for n' the number of vertices in $\mathcal{T}[u, u']$.*

The proofs can be found in the full version. We obtain:

Theorem 4. *Given a 1.5D terrain $\mathcal{T}$, the Voronoi visibility map $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$ can be computed in $O(n + (m^2 + k_c) \log n + k_v(m + \log n \log m))$ time.*

3 2.5D Terrains

3.1 Complexity of the Visibility Structures

Proposition 1. *The visibility map $\mathrm{Vis}(\mathcal{T}, P)$ of a 2.5D terrain $\mathcal{T}$ can have complexity $\Omega(m^2 n^2)$.*

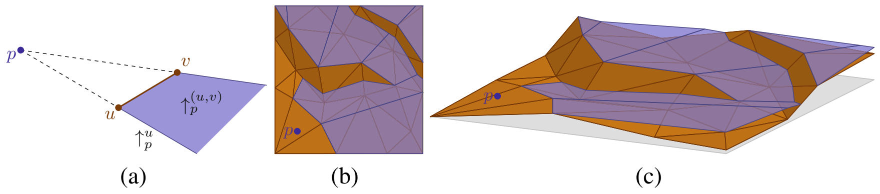


Fig. 5. (a) A ray and a vase. (b) The top-down view of a terrain $\mathcal{T}$ with a single viewpoint p. The domain is decomposed in the viewshed $\mathrm{Vis}(\mathcal{T}, p)$ and a collection of vases. (c) a 3D view of $\mathcal{T}$ and the vases of p.

Proof. We present a terrain that consists of a flat (horizontal) rectangle, the *courtyard*, surrounded by a thin wall. We make $O(n)$ (almost) vertical incisions, or *windows*, in the northern and western wall. We place half our viewpoints behind windows in the northern wall, and the other half behind windows in the western wall. Each viewpoint is placed so that it can see through $O(n)$ windows into the courtyard, see Fig. 4. It follows that the visibility map inside the courtyard forms an $O(mn) \times O(mn)$ grid. □

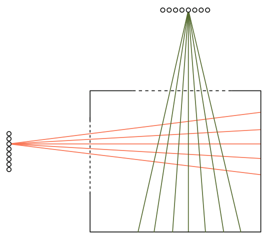


Fig. 4. Viewpoints are shown as white circles and rays indicate the part of the terrain visible from the viewpoint

In order to establish an upper bound on the complexity of the visibility maps, we start with the most general case, in which $\mathcal{T}$ is actually an arbitrary polyhedron.

Let $\mathcal{M}$ be a polyhedron, let v be a vertex of $\mathcal{M}$, and let $p \in \mathcal{P}$ be a viewpoint. We define the *ray* of p and v, denoted $\uparrow_p^v$, to be the half line that starts at v and has vector $\overrightarrow{pv}$. Similarly, let $p \in \mathcal{P}$ be a point and $e = \overline{uv}$ be an edge of $\mathcal{M}$. The *vase* of p and e, denoted $\uparrow_p^e$, is the region in $\mathbb{R}^3$ bounded by e, $\uparrow_p^u$, and $\uparrow_p^v$ (see Fig. 5(a)). The set of all vases originating from p is denoted $\Uparrow(p) = \{\uparrow_p^e | \ e \in E(\mathcal{M})\}$. Assuming general position, we have:

Observation 2. $\mathrm{Vis}(\mathcal{M}, \mathcal{P})$ *can have three types of vertices: (1) vertices of* $\mathcal{M}$*, (2) intersections between an edge of* $\mathcal{M}$ *and a vase, and (3) intersections between a triangle of* $\mathcal{M}$ *and two vases.*[1]

Theorem 5. *The visibility map* $\mathrm{Vis}(\mathcal{M}, \mathcal{P})$ *of a polyhedron* $\mathcal{M}$ *has complexity* $O(m^2n^3)$.

Proof. Each vase comes from a viewpoint in $\mathcal{P}$ and an edge in $E(\mathcal{M})$. Clearly, $|V(\mathcal{M})|$, $|E(\mathcal{M})|$, $|F(\mathcal{M})| \in O(n)$. So, the number of vertices of type (1), (2), and (3) is at most $O(n)$, $O(mn^2)$, and $O(m^2n^3)$, respectively. □

Next, we show that if $\mathcal{M}$ is a terrain, then the number of vertices of type (3) can only be $O(m^3n^2)$. Given any object $B \in \mathbb{R}^3$, we denote by $\underline{B}$ the vertical projection of B to $\mathbb{R}^2$. Furthermore, we define $S_1 \oplus S_2$ to be the overlay of subdivisions S_1 and S_2. Let $\uparrow_s^e$

[1] It is worth noting that there is a fourth possible type of vertex: an intersection of three vases. However, such a vertex does not lie on $\mathcal{M}$, so it does not appear in the visibility map.

and $\uparrow_t^f$ be two vases. The intersection of these two vases is a line segment (or half line), which we denote by ${}_s^e\!\times_t^f$. We call this a *pyramid ray*.

Observation 3. *Consider k planar subdivisions $S_1, S_2, \ldots, S_k$, and let $\mathbb{S} = \bigoplus_{i=1}^k S_i$ be their overlay. Any line ℓ has at most $O(\sum_{i=1}^k |S_i|)$ intersections with $\mathbb{S}$.*

Lemma 3. *Let R be the set of pyramid rays created by $\mathcal{P}$ on a 2.5D Terrain $\mathcal{T}$. Every edge $\underline{e} \in \underline{E(\mathcal{T})}$ intersects at most $O(m^3 n)$ rays from $\underline{R}$.*

Proof. Let $\underline{X}_i$ be the subdivision of $\mathbb{R}^2$ that we obtain by vertically projecting the upper envelope of $\mathcal{T}$ and all vases in $\Uparrow(p_i)$. Fig. 5(b) shows an example. Any pyramid ray $r \in R$ is the intersection of one vase from $\Uparrow(p_i)$ and one vase from $\Uparrow(p_j)$, for some $i \neq j$. This means that $\underline{r}$ is contained in a cell of $\underline{X}_i \oplus \underline{X}_j$. Let $\mathbb{X} = \bigoplus_{i=1}^m \underline{X}_i$. Then each cell in $\mathbb{X}$ is contained in at most m different projected vases, hence, it contains at most $\binom{m}{2}$ (pieces of) projected pyramid rays.

There are $O(n)$ vases in $\Uparrow(p_i)$, so $\underline{X}_i$ has $O(n)$ vertices. From Obs. 3 it follows that any line —and therefore any edge $\underline{e} \in \underline{E(\mathcal{T})}$— intersects the edges of $\mathbb{X}$ at most $O(mn)$ times. This means $\underline{e}$ intersects at most $O(mn)$ cells, and therefore also at most $O(m^3 n)$ pyramid rays in $\underline{R}$. □

Lemma 4. $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ *contains at most $O(m^3 n^2)$ vertices of type (3).*

Proof. We split the vertices of $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ of type (3) into two subtypes. Each vertex v (of type (3)) is associated with one pyramid ray r. Now, v is either of type (3)a, if it is the highest vertex on r, or of type (3)b otherwise. The number of vertices of type (3)a is at most $O(m^2 n^2)$, since there is at most one per ray and there are only $O(m^2 n^2)$ rays. We now show the number of vertices of type (3)b is at most $O(m^3 n^2)$.

Let v be a vertex of $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ of type (3)b. It is the intersection of a ray r and a triangle $t \in F(\mathcal{T})$. Since v is not the highest vertex on r, there must be another vertex w on r. Clearly, w cannot lie on t, so $\underline{w}$ must lie outside $\underline{t}$, while $\underline{v}$ lies inside $\underline{t}$. Thus there must be an edge $e \in E(P)$ such that $\underline{r}$ crosses $\underline{e}$. We charge v to this intersection between $\underline{r}$ and $\underline{e}$. Clearly, any such intersection gets charged at most once. By Lemma 3, there are at most $O(m^3 n^2)$ such intersections in total. Hence, the number of vertices of type (3)b is also at most $O(m^3 n^2)$. □

Theorem 6. $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$*, for $\mathcal{T}$ a 2.5D terrain, has complexity $O(m^3 n^2)$.*

The visibility map $\mathrm{Vis}(\mathcal{T}, \mathcal{P})$ corresponds to the union over $\mathcal{P}$ of the viewsheds of the individual viewpoints. Similarly, the colored visibility map corresponds to the overlay of the viewsheds of the individual viewpoints in $\mathcal{P}$. Therefore, Obs. 2 also holds for the vertices of $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$. This implies the following result.

Theorem 7. $\mathrm{ColVis}(\mathcal{T}, \mathcal{P})$*, for $\mathcal{T}$ a 2.5D terrain, has complexity $O(m^3 n^2)$.*

Finally, we are interested in the Voronoi visibility map. $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$ can have additional types of vertices: intersections of Voronoi edges with terrain triangles. We use *power diagrams*: Let $\mathcal{C} = C_1, .., C_m$ be a set of m circles in $\mathbb{R}^2$, and let c_i and r_i denote the center and radius of C_i, respectively. The (2D) power diagram $\text{PD}(\mathcal{C})$ is the subdivision of $\mathbb{R}^2$ into m regions, one for each circle, such that $R_i = \{x \in \mathbb{R}^2$ s. t., for all $j \in \{1,..,m\}, pow(C_i,x) \leq pow(C_j,x)\}$, where $pow(C_i,x) = d_2(c_i,x)^2 - r_i^2$ (and $d_2(\cdot,\cdot)$ denotes the Euclidean distance in $\mathbb{R}^2$). The (2D) power diagram of m circles has complexity $O(m)$ and can be computed in $O(m \log m)$ time [1].

Let $\text{VD}(\mathcal{P})$ denote the 3-dimensional Voronoi diagram of $\mathcal{P}$. We observe that the restriction of $\text{VD}(\mathcal{P})$ to any single plane H in $\mathbb{R}^3$ corresponds to a *power diagram* $\text{PD}(\mathcal{C}_\mathcal{P})$ in $\mathbb{R}^2$: Assume without loss of generality that H is a horizontal plane at $z = 0$, and let $\xi \geq \max_{p\in\mathcal{P}} p_z^2$ be some large value. Any point $a \in H$ is closer to $p \in \mathcal{P}$ than to $q \in \mathcal{P}$ if (and only if) $d(a,p) = d_3(a,p) \leq d_3(a,q)$, and hence if $d_3(a,p)^2 \leq d_3(a,q)^2$. Using that $a_z = 0$ we can rewrite this to $d_2(a,\underline{p})^2 - (\xi - p_z^2) \leq d_2(a,\underline{q}) - (\xi - q_z^2)$. So if we introduce a circle C_p in $\mathcal{C}_\mathcal{P}$ for every viewpoint p with center $\underline{p}$ and radius r_p such that $r_p^2 = \xi - p_z^2$ then we get that a is closer to p than to q if and only if $pow(C_p,a) \leq pow(C_q,a)$. Thus, we can prove:

Theorem 8. $\text{VorVis}(\mathcal{T},\mathcal{P})$*, for* $\mathcal{T}$ *a 2.5D terrain, has complexity* $O(m^4n^2)$*.*

3.2 Algorithms to Construct the Visibility Structures

Computing the (Colored) Visibility Map. Katz et al. [9] developed an $O((n\alpha(n) + k)\log n)$ time algorithm to compute the viewshed of a single viewpoint, where k is the output complexity and $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function. Coll et al. [6] use this algorithm to compute the visibility map of a 2.5D terrain in $O(m^2n^4)$ time and space. Essentially they project the individual viewsheds onto $\mathbb{R}^2$, and construct the overlay $\mathbb{V} = \bigoplus_{p\in\mathcal{P}} \underline{\mathcal{V}_\mathcal{T}(p)}$ (see Fig. 6). It is then easy to construct the (colored) visibility map from $\mathbb{V}$. We use the same approach. However, using our observations from the previous section, we show that even if the viewsheds have complexity $\Theta(n^2)$, we can compute the (colored) visibility map in $O(m^4n^2 \log n)$ time.

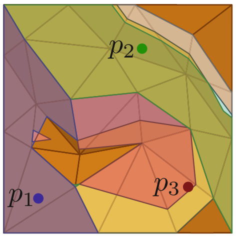


Fig. 6. Overlay $\mathbb{V}$

Lemma 5. *Given a 2.5D terrain* $\mathcal{T}$ *with* n *vertices and a set* $\mathcal{P}$ *of* m *viewpoints. The planar subdivision* $\mathbb{V}$ *can be constructed in* $O(m(n\alpha(n) + k_c)\log n)$ *time.*

Theorem 9. *Both the visibility* $\text{Vis}(\mathcal{T},\mathcal{P})$ *and the colored visibility map* $\text{ColVis}(\mathcal{T},\mathcal{P})$*, for* $\mathcal{T}$ *a 2.5D terrain can be computed in* $O(m(n\alpha(n) + k_c)\log n)$ *time.*

Computing the Voronoi Visibility Map. Let F be a face of the colored visibility map $\text{ColVis}(\mathcal{T},\mathcal{P})$, and let $\mathcal{P}_F$ denote the set of viewpoints that can see F. For each such face F we compute the intersection of F with the $\text{VD}(\mathcal{P}_F)$. We do this via the power diagram: i.e. consider the plane H containing F, and compute the power diagram on H with respect to the the viewpoints in $\mathcal{P}_F$. This takes $O(k_c m \log m)$ time in total, since $\text{ColVis}(\mathcal{T},\mathcal{P})$ has $O(k_c)$ faces, and each power diagram can be computed in $O(m \log m)$ time. Each power diagram is constrained to a single face, so we glue all of them together and project the result onto $\mathbb{R}^2$. This yields a subdivision $\mathbb{W}$ of size $O(k_c m)$. We now compute $\mathbb{V}$ in $O(m(n\alpha(n) + k_c)\log n)$ time (as described above), and overlay it with $\mathbb{W}$ in $O(k_c m + k_c + k_v) = O(k_c m)$ time. Hence:

Theorem 10. *The Voronoi visibility map* $\mathrm{VorVis}(\mathcal{T}, \mathcal{P})$*, for* $\mathcal{T}$ *a 2.5D terrain, can be computed in* $O(m(n\alpha(n) + k_c)\log n)$ *time.*

4 Final Remarks

We studied visibility with multiple viewpoints on polyhedral terrains for the first time. Our results show that considering multiple viewpoints converts classical visibility problems into much more challenging ones, even for 1.5D terrains.

Moreover, our results lead to many intriguing questions. For 1.5D terrains, is there an efficient algorithm to construct the Voronoi visibility map whose running time does not depend on k_c? In 2.5D, the worst-case complexities are not tight; it would be interesting to close those gaps. Algorithmically, in 2.5D the main challenge is to find an algorithm to construct the structures directly, avoiding the computation of the individual viewsheds. Finally, an interesting and realistic extension is when viewpoints have *limited* sight (i.e. can only see up to a certain distance). We discuss this extensively in the full version.

References

1. Aurenhammer, F.: Power diagrams: Properties, algorithms and applications. SIAM J. Comput. 16(1), 78–96 (1987)
2. Ben-Moshe, B., Katz, M., Mitchell, J.: A constant-factor approximation algorithm for optimal 1.5D terrain guarding. SIAM J. Comput. 36(6), 1631–1647 (2007)
3. Bose, P., Shermer, T., Toussaint, G., Zhu, B.: Guarding polyhedral terrains. Comput. Geom. 7(3), 173–185 (1997)
4. Catry, F., Rego, F., Santos, T., Almeida, J., Relvas, P.: Fires prevention in Portugal - using GIS to help improving early fire detection effectiveness. In: 4th Int. Wildland Fire Conf. (2007)
5. Cole, R., Sharir, M.: Visibility problems for polyhedral terrains. J. Symbolic Comput. 7(1), 11–30 (1989)
6. Coll, N., Madern, N., Sellarès, J.A.: Good-visibility maps visualization. Vis. Comput. 26(2), 109–120 (2010)
7. Gibson, M., Kanade, G., Krohn, E., Varadarajan, K.: An approximation scheme for terrain guarding. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 140–148. Springer, Heidelberg (2009)
8. Hurtado, F., Löffler, M., Matos, I., Sacristán, V., Saumell, M., Silveira, R.I., Staals, F.: Terrain visibility with multiple viewpoints (2013), arXiv:1309.4323 [cs.CG]
9. Katz, M.J., Overmars, M.H., Sharir, M.: Efficient hidden surface removal for objects with small union size. Comput. Geom. 2, 223–234 (1992)
10. King, J., Krohn, E.: Terrain guarding is NP-hard. SIAM J. Comput. 40(5), 1316–1339 (2011)
11. Lv, P., Zhang, J.-F., Lu, M.: An optimal method for multiple observers sitting on terrain based on improved simulated annealing techniques. In: Ali, M., Dapoigny, R. (eds.) IEA/AIE 2006. LNCS (LNAI), vol. 4031, pp. 373–382. Springer, Heidelberg (2006)
12. Möller, B.: Changing wind-power landscapes: regional assessment of visual impact on land use and population in Northern Jutland, Denmark. Appl. Energ. 83(5), 477–494 (2006)
13. Reif, J.H., Sen, S.: An efficient output-sensitive hidden surface removal algorithm and its parallelization. In: Proc. 4th Symp. Computational Geometry, pp. 193–200. ACM (1988)
14. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. 52(12), 2292–2330 (2008)

Exact Algorithms for Maximum Independent Set

Mingyu Xiao[1,⋆] and Hiroshi Nagamochi[2]

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

Abstract. We show that the maximum independent set problem (MIS) on an n-vertex graph can be solved in $1.2002^n n^{O(1)}$ time and polynomial space, which is even faster than Robson's $1.2109^n n^{O(1)}$-time exponential-space algorithm published in 1986. We also obtain improved algorithms for MIS in graphs with maximum degree 6 and 7. Our algorithms are obtained by effectively using fast algorithms for MIS in low-degree graphs and making careful analyses for MIS in high-degree graphs.

1 Introduction

Over the last few decades, an extensive research has been done on exact exponential algorithms. Many interesting methods and results have been obtained in this area, which can be found in a nice survey by Woeginger [14] and a recent monograph by Fomin and Kratsch [5]. In the line of research on worst-case analysis of exact algorithms for NP-hard problems, the *maximum independent set* problem (MIS) is undoubtedly one of the most fundamental problems. The problem is used to test the efficiency of some new techniques of exact algorithms and often introduced as the first problem in some textbooks and lecture notes of exact algorithms. However, despite of a large number of contributions on exact algorithms and their worst-case analyses for MIS during the last 30 years, no published algorithm runs faster than the $1.2109^n n^{O(1)}$-time exponential-space algorithm by Robson in 1986 [10]. Fomin and Kratsch say that 'the running time of current branching algorithms for MIS with more and more detailed analyses seems to converge somewhere near 1.2^n' [5]. Researchers are interesting in how fast we can exactly solve MIS.

Related Work. The first nontrivial exact algorithm for MIS is back to Tarjan and Trojanowski's $2^{n/3} n^{O(1)}$-time algorithm in 1977 [12]. Later, Jian obtained a $1.2346^n n^{O(1)}$-time algorithm [7]. Robson gave a $1.2278^n n^{O(1)}$-time polynomial-space algorithm and a $1.2109^n n^{O(1)}$-time exponential-space algorithm [10]. Robson also claimed better running times in a technical report [11]. Fomin *et al.* [4] introduced the "measure-and-conquer" method and got a simple $1.2210^n n^{O(1)}$-time polynomial-space algorithm by using this method. Also based on this

⋆ Supported by NFSC of China under the Grant 61370071 and Fundamental Research Funds for the Central Universities under the Grant ZYGX2012J069.

method, Kneis *et al.* [8] and Bourgeois *et a*l. [1] improved the running time bound to $1.2132^n n^{O(1)}$ and $1.2114^n n^{O(1)}$ respectively, which are the current fastest polynomial-space algorithms for MIS in published articles. There is also a large amount of contributions to MIS in degree-bounded graphs [9,6,15,16,17,18]. Now MIS-3 (MIS-i means MIS in graphs with maximum degree i) can be solved in $1.0836^n n^{O(1)}$ time [16], MIS-4 can be solved in $1.1376^n n^{O(1)}$ time [17], MIS-5 can be solved in $1.1737^n n^{O(1)}$ time [18] and MIS-6 can be solved in $1.2050^n n^{O(1)}$ time [1], where all of them use only polynomial space. The measure-and-conquer method is a very powerful tool to design or analyze exact algorithms. Most fast polynomial-space algorithms for MIS are designed based on the method. By combining this method with a bottom-up method, Bourgeois *et al.* [1] got the $1.2114^n n^{O(1)}$-time polynomial-space algorithm for MIS. Their algorithm is based on fast algorithms for MIS in low-degree graphs.

Our Contributions. In this paper, we will design an $1.2002^n n^{O(1)}$-time polynomial-space algorithm for MIS, which is faster than Robson's $1.2109^n n^{O(1)}$-time exponential-space algorithm [10] obtained in 1986. We also show that MIS-6 and MIS-7 can be solved in $1.1898^n n^{O(1)}$ and $1.1976^n n^{O(1)}$ time, respectively. Our algorithms use the measure-and-conquer method. But the improvement is not obtained by studying more cases in previous algorithms. Instead, we will introduce some new methods to reduce a large number of cases and make the algorithm and its analysis easy to follow. Our algorithm also needs to use our previous fast algorithms for MIS in low-degree graphs. The improvement is mainly obtained by using the following ideas:

1. We introduce a divide-and-conquer method to get improved algorithms for MIS in high-degree graphs based on fast algorithms for MIS in low-degree graphs. In the method, we separate the algorithm into two parts. One part is to deal with the problem in graphs with maximum degree at most i. The other part is to effectively deal with vertices of degree $> i$ in the graph. We may use this idea to design fast algorithms for MIS in degree bound graphs. Once an algorithm for MIS-i is obtained, we design algorithms for MIS-$(i + 1)$ based on it. Similar bottom-up ideas have been used in some previous algorithms, such as the algorithm for MIS in [1] and the algorithm for the parameterized vertex cover problem in [2]. One advantage of our method is that, the divide-and-conquer method can combine the measure-and-conquer method well to design exact algorithms. Then we can catch the properties of fast algorithms for MIS in low-degree graphs and propagates the improvement from instances of low-degree graphs to instances of high-degree graphs.
2. We also introduce a method to reduce a huge number of case analyses in the algorithms and then our algorithms become much easier to check the correctness. This method is based on Lemma 4 in Section 4. It can also be directly used to reduce a large number of cases in the analysis of previous algorithms without modifying the algorithms.
3. We introduce a new branching rule, called "branching on edges," to deal with vertices of degree $> i$ in some dense local structures, where it may lead to a bad performance if we just branch on a vertex of maximum degree to search a solution.

2 Preliminaries

2.1 Notation System

Let $G = (V, E)$ be a graph and $n = |V|$ be the number of vertices in G. We will use n_i to denote the number vertices of degree i in G, and $\alpha(G)$ to denote the size of a maximum independent set of G. For a subset $X \subseteq V$, let $N(X)$ denote the set of all vertices in $V - X$ that are adjacent to a vertex in X, and $N[X] = X \cup N(X)$. For a vertex v, let $\delta(v) = |N(v)|$ denote the degree of a vertex v, and $N_2(v)$ denote the set of vertices with distance exactly 2 from v, and let $N_2[v] = N_2(v) \cup N[v]$. For a vertex v and its neighbor $u \in N(v)$, a vertex $z \in N_2(v)$ adjacent to u is called an *outer-neighbor* of u at v, and the *outer-degree* of u at v is the number of outer-neighbors of u at v. For a vertex v, let f_v denote the number of edges between $N[v]$ and $N_2(v)$, and let e_v denote the number of edges in the graph $G[N(v)]$ induced by the neighbors of v. Let $k_v = (k_1, k_2, \ldots, k_d)$ denote the sequence of the number k_i of degree-i neighbors of a vertex v, where we may denote $k_v = (k_3, k_4, k_5, k_6)$ when v has no neighbor of degree $i \notin \{3, 4, 5, 6\}$. For a vertex set $X \subseteq V$, let $G - X$ be the graph obtained from G by removing X together with edges incident on any vertex in X, and G/X denote the graph obtained from G by contracting X into a single vertex (removing self-loops and parallel edges).

2.2 Branching Algorithms and the Measure-and-Conquer Method

Our algorithms use a branch-and-reduce paradigm. In MIS, a branching rule will branch on the current instance G into several instances $G_1, G_2, \ldots, G_l$ such that the measure μ_i of each G_i is less than the measure μ of G, and a solution to G can be found in polynomial time if a solution to each of the l instances $G_1, G_2, \ldots, G_l$ is known. We will use $C(\mu)$ to denote the worst-case size of the search tree in the algorithm when the measure of the instance is at most μ. The above branch creates the recurrence relation $C(\mu) \leq \sum_{i=1}^{l} C(\mu - \mu'_i)$, where $\mu'_i = \mu - \mu_i$. The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-\mu'_i}$, denoted by $\tau(\mu'_1, \mu'_2, \ldots, \mu'_l)$, is also called the *branching factor* of the above recurrence relation. Let τ be the maximum branching factor among all branching factors in the search tree. Then the size of the search tree $C(\mu) = O(\tau^{\mu})$. More details about the analysis and how to solve recurrences can be found in the monograph [5].

To reduce the size of the search tree, we may need to design good branching rules for the algorithm. The selection of the measure is also an important issue before designing the branching rules. The measure-and-conquer method [4] allows us to define a sophisticated measure. In this method, we set a weight to each vertex in the graph according to the degree of the vertex and define the sum of the weights in the graph to be the measure. An important step in this method is to set a vertex weight. We then need to solve a quasiconvex program to get a good weight setting. In this paper we also use the branch-and-reduce paradigm and the measure-and-conquer method to design our algorithm.

2.3 Reduction Operations

Before applying our branching rules, we may first apply reduction rules to reduce some local structures. Many nice reduction rules have been developed. In this paper, we only use three known reduction rules.

Reduction by Removing Unconfined Vertices
A vertex v in an instance G is called *removable* if $\alpha(G) = \alpha(G - v)$. A sufficient condition for a vertex to be removable has been studied in [16]. In this paper, we only use a simple case of the condition. A neighbor $u \in N(v)$ of v is called an *extending child* of v if u has exactly one outer-neighbor $s_u \in N_2(v)$ at v, where s_u is also called an *extending grandchild* of v. Let $N^*(v)$ denote the set of all extending children $u \in N(v)$ of v, and S_v be the set of all extending grandchildren s_u $(u \in N^*(v))$ of v together with v itself. We call v *unconfined* if there is a neighbor $u \in N(v)$ which has no outer-neighbor or $S_v \setminus \{v\}$ is not an independent set (i.e., some two vertices in $S_v \cap N_2(v)$ are adjacent) [1]. It is known in [16] that any unconfined vertex is removable.

Lemma 1. [16] *For an unconfined vertex v in graph G, it holds that*

$$\alpha(G) = \alpha(G - v).$$

A vertex u *dominates* another vertex v if $N[u] \subseteq N[v]$, where v is called *dominated*. We see that dominated vertices are unconfined vertices.

Reduction by Folding Complete k-independent Sets
We call a set $A = \{v_1, \ldots, v_k\}$ of k degree-$(k + 1)$ vertices a *complete k-independent set* if they have common neighbors $N(v_1) = \cdots = N(v_k)$.

Lemma 2. [16] *For a complete k-independent set A, we have that*

$$\alpha(G) = \alpha(G^\star) + k,$$

where $G^\star = G/N[A]$ if $N(A)$ is an independent set and $G^\star = G - N[A]$ otherwise.

Folding a complete k-independent set A is to eliminate the set $N[A]$ from an instance in the above way. In our algorithm, we only fold complete k-independent set with $k \leq 2$, since this operation is good enough for our analysis. Folding a complete 1-independent set $A = \{v\}$ consisting of a degree-2 vertex v is also called *folding a degree-2 vertex v*.

Reduction by Removing Line Graphs
If a graph H is the line graph of a graph H', then a maximum independent set H can be obtained as the set of vertices that corresponds the set of edges in a maximum matching in H'. To reduce a worst case, we need to remove the line graphs of 4-regular graphs in our algorithm for MIS-6. A graph is the line graph of a 4-regular graph if and only if the graph has only degree-6 vertices and each of them is contained in two edge-disjoint cliques of size 4. More characterizations of line graphs can be found in [13].

[1] Unconfined vertices in [16] are defined in a more general way.

Definition 1. *A graph is called a* reduced graph*, if it contains none of unconfined vertices, complete k-independent sets with $k = 1$ or 2, and a component of a line graph of a 4-regular graph .*

When a graph is not a reduced graph, we can apply the reduction rules in this section to reduce the graph. We will simply use $s := \mathrm{RG}(G)$ to denote the processing of all above reduction operations that returns s as the size of the partial solution found by it.

3 The Divide-and-Conquer Method

We will use a divide-and-conquer method to solve MIS. In this method, we split MIS into two subproblems according to an integer $i > 3$, and design algorithms for the two parts separately. The first part is to design a fast algorithm for MIS-i and the second part is to design effective branching rules for the graph with at least one vertex of degree $> i$. We can combine these two parts to get an algorithm for MIS in this way: use the branching rules in the second part to search a solution if the graph has a vertex of degree $> i$, and call the algorithm for MIS-i in the first part directly otherwise. However, sometimes it is not easy to analyze the combined running time, since that the measures used in the two parts may be different. For example, if we use the measure-and-conquer method to analyze algorithms (or branching rules), we may set different vertex weights (and then the measures may be different). We will introduce a method to effectively deal with this trouble, especially for the case where the measure is set as the sum of total weight of vertices in the graph.

Let A_i denote an algorithm that solves MIS-i in a graph G of maximum degree $\leq i$ in $(\tau_i)^{\mu_i(G)}|G|^{O(1)}$ time, where τ_i is a positive number and $\mu_i(G) = \sum_{1 \leq j \leq i} w_j^{\langle i \rangle} n_j$ is the measure of G (recall that n_j is the number of degree-j vertices in G and $w_j^{\langle i \rangle} \geq 0$ is the weight of a degree-j vertex). Let $B_{>i}$ denote a procedure that branches on a graph G of maximum degree $> i$ with branching factor τ_i' on measure $\mu_{i+1}(G) = \sum_{j \geq 1} w_j^{\langle i+1 \rangle} n_j$, where $w_j^{\langle i+1 \rangle} \geq 0$ is the weight of a degree-j vertex in the procedure. We have the following lemma:

Lemma 3. *For an integer $i \geq 3$, let $\lambda = \max\{\frac{w_j^{\langle i \rangle}}{w_j^{\langle i+1 \rangle}} \mid 0 \leq j \leq i,\ w_j^{\langle i+1 \rangle} \neq 0\}$ and $\tau_{i+1} = \max\{\tau_i', (\tau_i)^{\lambda}\}$. Then MIS can be solved in $(\tau_{i+1})^{\mu_{i+1}(G)}|G|^{O(1)}$ time.*

A proof of Lemma 3 can be found in the full version of this paper.

The above divide-and-conquer method can also be used to design algorithms for MIS in degree bounded graphs (not only for MIS in general graphs). For example, to design an algorithm for MIS-$(i+1)$, we can first solve MIS-i and then consider branching rules for graphs with maximum degree $> i$. The framework of the analysis is the same.

Here is an application of Lemma 3. We will show that MIS-8 can be solved in time $1.2002^{\mu_8(G)}|G|^{O(1)}$ time, where $\mu_8(G) = 0.6483n_3 + 0.8519n_4 + 0.9077n_5 + 0.9557n_6 + 0.9880n_7 + n_8$, and that in a graph with maximum degree at least 9 we can branch with branching factor 1.1975 on the measure $\mu_9(G) = \sum_j n_j$. In

Lemma 3, we have $\tau'_8 = 1.1975$, $\tau_8 = 1.2002$, and $\lambda = \max\{0.6483, 0.8519, 0.9077, 0.9557, 0.9880, 1\} = 1$. Then MIS can be solved in $1.2002^n n^{O(1)}$ time.

In the above method, we let $\tau_{i+1} = \max\{\tau'_i, (\tau_i)^\lambda\}$, where τ'_i is decided by $B_{>i}$, τ_i is decided by A_i, and λ is related to the vertex weights in both of $B_{>i}$ and A_i. So sometimes simple reductions on τ'_i or τ_i may not lead to improvement on the algorithm A_{i+1}. To get more properties and further improvements on the problem, in our algorithm, we may not design A_i and $B_{>i}$ totally independently. Instead, we will design $B_{>i}$ based on A_i by considering the result (the values of τ_i and vertex weight) of A_i as some constraints to set the vertex weight in $B_{>i}$.

This divide-and-conquer method provides a way to solve MIS by solving two subproblems and to design fast algorithms for MIS based on fast algorithms for MIS in low-degree graphs. We will focus on the subalgorithm $B_{>i}$. Fast algorithms A_i for MIS-i with $i = 3, 4$ and 5 can be found in references [16,17,18].

In this paper, by using this divide-and-conquer method, first, we design an algorithm for MIS-6 based on fast algorithm for MIS-5 in [18], second, we design an algorithm for MIS-7 based on the algorithm for MIS-6, third, we design an algorithm for MIS-8 based on the algorithm for MIS-7, and finally, we design an algorithm for MIS in general graphs based on the algorithm for MIS-8. Our results are listed in Table 1.

Table 1. Our algorithms designed by the divide-and-conquer method

Problems	Running time	The vertex weight
MIS-6	$1.1898^n n^{O(1)}$	$(w_3, w_4, w_5, w_6) = (0.5139, 0.7632, 0.9215, 1)$
MIS-7	$1.1976^n n^{O(1)}$	$(w_3, w_4, w_5, w_6, w_7) =$ (0.5781, 0.7958, 0.8882, 0.9639,1)
MIS-8 (MIS)	$1.2002^n n^{O(1)}$	$(w_3, w_4, w_5, w_6, w_7, w_8) =$ (0.6483,0.8519,0.9077,0.9557,0.9880,1)

4 Branching on High-Degree Vertices

We can simply branch on a high-degree vertex v into two branches by including it to the solution set or not. In the branch where v is included to the solution, $N[v]$ will be deleted from the graph since the neighbors of v cannot be selected into the solution anymore. If the degree of v is higher, then the graph can be reduced more in this branch. We extend the simple branch rule based on this following observation. For a vertex v, there are only two possible cases: (i) there is a maximum independent set of the graph which does not contain v; and (ii) every maximum independent set of the graph contains v. It is shown in [16] that for Case (ii), S_v is always contained in any maximum independent set of the graph (recall that S_v is the set of all extending grandchildren of v together with v itself). We get the following branching rule.

Branching on a vertex v means generating two subinstances by excluding v from the independent set or including S_v to the independent set. In the first branch we will delete v from the instance whereas in the second branch we will delete $N[S_v]$ from the instance.

Branching on a vertex v of maximum degree d is one of the most fundamental operations in our algorithm. We analyze this operation. Let $\Delta_{out}(v)$ and $\Delta_{in}(v)$ to denote the decrease of the measure of μ in the branches of excluding v and including S_v, respectively. We get recurrence $C(\mu) = C(\mu - \Delta_{out}(v)) + C(\mu - \Delta_{in}(v))$. We give more details about lower bounds on $\Delta_{out}(v)$ and $\Delta_{in}(v)$. Let k_i denote the number of degree-i neighbors of v. Then $d = \sum_{i=3}^{d} k_i$. For the first branch, we get

$$\Delta_{out}(v) = w_d + \sum_{i=3}^{d} k_i \Delta w_i,$$

where $\Delta w_i = w_i - w_{i-1}$.

In the second branch, we will delete $N[S_v]$ from the graph. Let $\Delta(\overline{N[v]})$ denote the decrease of weight of vertices in $V(G) - N[v]$ by removing $N[S_v]$ from G together with possibly weight decrease attained by reduction operations applied to $G - N[S_v]$. Then we have

$$\Delta_{in}(v) \geq w_d + \sum_{i=3}^{d} k_i w_i + \Delta(\overline{N[v]}).$$

We can branch on a vertex v of degree d with recurrence

$$\begin{aligned} C(\mu) &= C(\mu - \Delta_{out}(v)) + C(\mu - \Delta_{in}(v)) \\ &\leq C(\mu - (w_d + \textstyle\sum_{i=3}^{d} k_i \Delta w_i)) + C(\mu - (w_d + \textstyle\sum_{i=3}^{d} k_i w_i + \Delta(\overline{N[v]}))). \end{aligned} \tag{1}$$

In our algorithm, we carefully select a vertex of maximum degree to branch on so that the branching factor from the worst recurrence (1) becomes as small as possible. To do so, we need to derive lower bounds on $\Delta(\overline{N[v]})$ and examine all possible configurations of $\{k_3, k_4, \ldots, k_d\}$.

We will discuss how to analyze and improve the lower bounds on $\Delta(\overline{N[v]})$ later. Now assume that a lower bound of $\Delta(\overline{N[v]})$ is fixed. In (1), each configuration of $\{k_3, k_4, \ldots, k_d\}$ will create a concrete recurrence. There are $(d+1)^{d-3}$ (the number of integer solutions to the function $k_3 + k_4 + \cdots + k_d = d$) configurations of $\{k_3, k_4, \ldots, k_d\}$. Then (1) will generate $(d+1)^{d-3}$ concrete recurrences. We will introduce a lemma to reduce the number of recurrences from $(d+1)^{d-3}$ to only $d-2$.

The following lemma can be used to eliminate redundant recurrences to determine the largest branching factor among a set of systematically generated recurrences.

Lemma 4. *Let $C(x) = \tau^x$ for a positive $\tau > 1$. For any nonnegative p, μ, a_i, b_i, $i = 1, 2, \ldots, \ell$ ($\ell \geq 1$), the maximum of*

$$C(\mu - (\sum_{i=1,2,\ldots,\ell} k_i a_i + c)) + C(\mu - (\sum_{i=1,2,\ldots,\ell} k_i b_i + d))$$

over all $k_1, k_2, \ldots, k_\ell \geq 0$ subject to $k_1 + k_2 + \cdots + k_\ell = p$ is equal to the maximum of

$$C(\mu - (pa_i + c)) + C(\mu - (pb_i + d))$$

over all $i = 1, 2, \ldots, \ell$.

Proof. It suffices to show that for nonnegative w, a_1, b_1, b_2, c, $d \geq 0$, it holds $C(\mu-(a_1+a_2+c)) + C(\mu-(b_1+b_2+d)) \leq \max\{C(\mu-(2a_1+c)) + C(\mu-(2b_1+d)), C(\mu-(2a_2+c)) + C(\mu-(2b_2+d))\}$. The lemma can be obtained by applying this repeatedly. Note that function $f(t) = \tau^{-(2a_1(1-t)+2a_2t+c-\mu)} + \tau^{-(2b_1(1-t)+2b_2t+d-\mu)}$ is convex since the second derivative is nonnegative. Hence $f(0.5) \leq \max\{f(0), f(1)\}$ holds, as required. ∎

By applying Lemma 4, in (1), we only need to consider $d-2$ concrete recurrences with $(k_3, k_4, \ldots, k_d) = (d, 0, \ldots, 0), (0, d, 0, \ldots, 0), \cdots, (0, \ldots, 0, d)$ respectively. For example, if $d = 6$, we can reduce the number of recurrences from $7^4 = 2401$ to only 5. Lemma 4 is introduced to simplify the analysis of recurrences for the first time. It can be used to reduced thousands of recurrences in the analysis of previous algorithms for MIS, such as the algorithms in [8] and [1]. Note that the authors of [8] introduce a computer-added method to create all possible recurrences in the web page [19]. There are more than 10 thousands recurrences listed. By using Lemma 4, we can reduce the number of recurrences to less than 50 and it becomes checkable by hand.

5 Branching on Edges

In some cases, $\Delta(\overline{N[v]})$ in (1) is small and the corresponding recurrences will become the bottlenecks of the algorithm. To increase $\Delta(\overline{N[v]})$, we can increase f_v (the number of edges between $N(v)$ and $N_2(v)$). When $N^*(v) \neq \emptyset$, we can remove some vertices in $N_2(v)$ and then decrease the measure by a large amount in $\overline{N[v]}$. Otherwise, $N^*(v) = \emptyset$ and then $f_v \geq 2\delta(v)$. To get further improvement on f_v, we introduce a new branching rule to deal with some dense local graphs.

Branching on Edges. Two disjoint independent subsets A and B of vertices in a graph G are called *alternative* if $|A| = |B| \geq 1$ and there is a maximum independent set S_G of G which satisfies $S_G \cap (A \cup B) = A$ or B. Let $G^\dagger$ be the graph obtained from G by removing $A \cup B \cup (N(A) \cap N(B))$ and adding an edge ab for every two nonadjacent vertices $a \in N(A) - N[B]$ and $b \in N(B) - N[A]$.

Lemma 5. [16] *For alternative subsets A and B in a graph G, $\alpha(G) = \alpha(G^\dagger) + |A|$.*

Lemma 6. *Let vv' be an edge. Then*

$$\alpha(G) = \max\{\alpha(G - \{v, v'\}), \alpha(G^\dagger) + 1\},$$

where $G^\dagger$ be the graph obtained from G by removing $\{v, v'\} \cup (N(v) \cap N(v'))$ and adding an edge ab for every two nonadjacent neighbors $a \in N(v) - N[v']$ and $b \in N(v') - N[v]$.

Proof. We easily observe that either (i) every maximum independent set S_G of G satisfies $S_G \cap \{v, v'\} = \emptyset$; or (ii) there is a maximum independent set S_G of G such that $S_G \cap \{v, v'\} \neq \emptyset$. In (i), we have $\alpha(G) = \alpha(G - \{v, v'\})$. In (ii), sets

$A = \{v\}$ and $B = \{v'\}$ are alternative in G, and we have $\alpha(G) = \alpha(G^\dagger) + 1$ by Lemma 5. ∎

Branching on an edge vv' means generating two subinstances according to Lemma 6. This is to either remove $\{v, v'\}$ from the graph G or construct the graph $G^\dagger$ from $G - (\{v, v'\} \cup (N(v) \cap N(v')))$ by making each pair $a \in N(v) - N[v']$ and $b \in N(v') - N[v]$ adjacent. Branching on an edge may not always be very effective. In our algorithms, we will apply it to edges vv' when $N(v) \cap N(v')$ is large, which are called "short edges."

The definitions of "short edges" in different degree-bounded graphs are slightly different. In a reduced graph of maximum degree 6, an edge vv' is called *short* if $\delta(v) = 6$, $\delta(v') \in \{5, 6\}$ and $|N(v) \cap N(v')| \geq 3$. In a reduced graph of maximum degree i_0 ($i_0 = 7$ or 8), an edge vv' is called *short* if $\delta(v) = \delta(v') = i_0$ and $|N(v) \cap N(v')| \geq i_0 - 3$. In a graph G of maximum degree 6, 7 or 8, a short edge is called *optimal* if $|N(v) \cap N(v')| - \delta(v')$ is maximized. In our algorithms, we only branch on optimal short edges in graphs of maximum degree 6, 7 and 8.

6 The Algorithms

We first give the algorithms for MIS-6, MIS-7 and MIS-8, then we discuss MIS in general graphs.

6.1 The Algorithms for MIS-6, MIS-7 and MIS-8

After dealing with short edges in a graph G of maximum degree $6, 7$ and 8, for each vertex v of maximum degree in G, we have that $f_v \geq 2|\delta(v)| + k_d$ ($d = \delta(v)$). In fact, the worst cases in our algorithms will be the cases of $k_{\delta(v)} = \delta(v)$, where $f_v \geq 2\delta(v) + k_{\delta(v)} = 3\delta(v)$. This inequality improves the bound of $2\delta(v)$ (before dealing with short edges) and leads improvements in our algorithms. We further avoid the worst case of $f_v = 3\delta(v)$ by branching on 'optimal vertices' of maximum degree (after branching on all short edges), and then we can get $f_v \geq 3\delta(v) + 2$ (or $f_v \geq 3\delta(v) + 6$ for graphs of maximum degree 8) for the cases of $k_{\delta(v)} = \delta(v)$. These cases will be the final worst cases in our algorithms.

In a reduced graph of maximum degree 6, a degree-6 vertex v is called *optimal* if at least one of the following (i)-(iv) is holds: (i) $N^*(v) \neq \emptyset$; (ii) the vertex v has at most four degree-6 neighbors ($k_6 \leq 4$); (iii) the vertex v has a neighbor of degree ≤ 4; and (iv) $f_v + (f_v - |N_2(v)|) + q_v \geq 8 + 2k_6$, where q_v is the number of vertices of degree < 6 in $N_2(v)$.

In a reduced graph of maximum degree i_0 ($i_0 = 7$ or 8), a degree-i_0 vertex v is called *optimal* if at least one of the following (i)-(iii) is holds: (i) $N^*(v) \neq \emptyset$; (ii) the vertex v has at most $i_0 - 2$ degree-i_0 neighbors ($k_{i_0} \leq i_0 - 2$); and (iii) $f_v + (f_v - |N_2(v)|) \geq 10 + 2k_{i_0}$ for $i_0 = 7$ and $f_v + (f_v - |N_2(v)|) \geq 14 + 2k_{i_0}$ for $i_0 = 8$.

Lemma 7. *In a reduced graph G of maximum degree 6, 7 or 8, if G has no short edges, then G has at least one optimal vertex.*

```
Input: A graph $G$.
Output: The size of a maximum independent set in $G$.

  1. Reduce the graph if it not a reduced graph (i.e., $(G, s) := \mathrm{RG}(G, 0)$), and
     let $d$ be the maximum degree of $G$.
  2. If $\{d \geq (i_0 + 1)\ \}$, pick up a vertex $v$ of maximum degree $d$, and return
     $s + \max\{\mathrm{MIS\_}i_0(G - v), |S_v| + \mathrm{MIS\_}i_0(G - N[S_v])\}$.
  3. Elseif$\{d = i_0$ and $G$ has a short edge$\}$, pick up an optimal short edge $vv'$,
     and return $s + \max\{\mathrm{MIS\_}i_0(G - \{v, v'\}), 1 + \mathrm{MIS\_}i_0(G^\dagger)\}$.
  4. Elseif $\{d = i_0$ ($G$ has no short edges)$\}$, pick up an optimal degree-$i_0$
     vertex $v$, and return $s + \max\{\mathrm{MIS\_}i_0(G - v), |S_v| + \mathrm{MIS\_}i_0(G - N[S_v])\}$.
  5. Else $\{G$ is a degree-$(i_0 - 1)$ graph$\}$, use our algorithm for MIS-$(i_0 - 1)$ to
     solve the instance $G$ and return $s + \alpha(G)$, where the algorithm for MIS-5
     is in [18].

Note: With a few modifications, the algorithm can deliver a maximum inde-
pendent set.
```

Fig. 1. Algorithms MIS_$i_0(G)$

A proof of Lemma 7 can be found in the full version of this paper. Our algorithms for MIS-i_0 ($i_0 = 6, 7$ or and 8) are described in Figure 1.

In our algorithms, we set the vertex weight as follows: for $0 \leq i \leq 2$, $w_i = 0$; for $3 \leq i \leq i_0$, w_i is set as that in Table 1; for $i \geq i_0+1$, $w_i = w_{i_0}+(i-i_0)(w_{i_0}-w_{i_0-1})$. We set vertex weight greater than 1 for vertices of degree $> i_0$ to simplify the analyses in our algorithms (recall that a vertex of degree $> i_0$ may be created after applying reduction rules in a graph of maximum degree i_0). This setting will not change the running time bound of our algorithms.

Lemma 8. *With the above vertex weight setting, each recurrence generated by the algorithm* MIS_6(G) (*resp.,* MIS_7(G), MIS_8(G)) *in Figure 1 has a branching factor not greater than* 1.1898 (*resp.,* 1.1976*,* 1.2002).

The proof of this analytical lemma can be found in the full version of this paper. Since the measure μ is not greater than the number n of vertices in the initial graph in MIS_6(G), MIS_7(G) and MIS_8(G), we get that

Theorem 1. *A maximum independent set in a degree-*6 *graph* (*resp., degree-*7 *graph, degree-*8 *graph*) *of n vertices can be found in* $1.1898^n n^{O(1)}$ (*resp.,* $1.1976^n n^{O(1)}$*,* $1.2002^n n^{O(1)}$) *time.*

6.2 MIS in General Graphs

The algorithm for MIS in general graphs in simple. It only contains two steps: branch on a vertex of maximum degree if the degree of the graph is at least 9, and call the algorithm for MIS-8 if the maximum degree of the graph is less than 9. In the subalgorithm dealing with vertices of degree ≥ 9, we set the measure as the number of vertices in the graph (the weight of each vertices is 1). Then we can get the following recurrence:

$$C(\mu) \leq C(\mu - 1) + C(\mu - 10), \tag{2}$$

which has branching factor 1.19749, better than 1.20018 for MIS-8. The analysis in Section 3 shows that MIS in general graphs can be solved in $1.2002^n n^{O(1)}$ time.

Theorem 2. *A maximum independent set in an n-vertex graph can be found in $1.2002^n n^{O(1)}$ time and polynomial space.*

References

1. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Fast algorithms for max independent set. Algorithmica 62(1-2), 382–415 (2012)
2. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science 411(40-42), 3736–3756 (2010)
3. Eppstein, D.: Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. ACM Transactions on Algorithms 2(4), 492–509 (2006)
4. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM 56(5), 1–32 (2009)
5. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer (2010)
6. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
7. Jian, T.: An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. IEEE Transactions on Computers 35(9), 847–851 (1986)
8. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: FSTTCS 2009. LIPIcs, vol. 4, pp. 287–298 (2009)
9. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with degree 3. J. of Disc. Alg. 7(2), 191–212 (2009)
10. Robson, J.: Algorithms for maximum independent sets. J. of Algorithms 7(3), 425–440 (1986)
11. Robson, J.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Univsersite Bordeaux I (2001)
12. Tarjan, R., Trojanowski, A.: Finding a maximum independent set. SIAM J. on Computing 6(3), 537–546 (1977)
13. West, D.: Introduction to Graph Theory. Prentice Hall (1996)
14. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)
15. Xiao, M.: A simple and fast algorithm for maximum independent set in 3-degree graphs. In: Rahman, M. S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 281–292. Springer, Heidelberg (2010)
16. Xiao, M., Nagamochi, H.: Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. Theoretical Computer Science 469, 92–104 (2013)
17. Xiao, M., Nagamochi, H.: A refined algorithm for maximum independent set in degree-4 graphs. Technical report 2013-002, Kyoto University (2013), `http://www.amp.i.kyoto-u.ac.jp/tecrep/abst/2013/2013-002.html`
18. Xiao, M., Nagamochi, H.: An exact algorithm for maximum independent set in degree-5 graphs. In: Fellows, M., Tan, X., Zhu, B. (eds.) FAW-AAIM 2013. LNCS, vol. 7924, pp. 72–83. Springer, Heidelberg (2013)
19. `http://www.tcs.rwth-aachen.de/independentset/`

On the Enumeration and Counting of Minimal Dominating sets in Interval and Permutation Graphs

Mamadou Moustapha Kanté[1], Vincent Limouzy[1], Arnaud Mary[1], Lhouari Nourine[1], and Takeaki Uno[2]

[1] Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
[2] National Institute of Informatics, Japan

Abstract. We reduce (in polynomial time) the enumeration of minimal dominating sets in interval and permutation graphs to the enumeration of paths in directed acyclic graphs(DAGs). As a consequence, we can enumerate with linear delay, after a polynomial time pre-processing, minimal dominating sets in interval and permutation graphs. We can also count them in polynomial time. This improves considerably upon previously known results on interval graphs, and up to our knowledge no output polynomial time algorithm for the enumeration of minimal dominating sets and their counting were known for permutation graphs.

1 Introduction

The Minimum Dominating Set problem is a classic and well-studied graph optimisation problem. A *dominating set* in a graph G is a subset D of its set of vertices such that each vertex is either in D or has a neighbour in D. Computing a minimum dominating set has numerous applications in many areas, *e.g.*, networks, graph theory (see for instance the book [9]). In this paper we are interested in the enumeration of (inclusion-wise) *minimal dominating sets* in interval and permutation graphs. The Minimum Dominating Set problem is NP-complete in general, but it is known to be solvable in linear time for these two classes [4,16].

There are two approaches in enumeration algorithms. The *input-sensitive* approach uses classical worst-case running time analysis, *i.e.*, the running time depends on the length of the input. This approach is usually used in exact exponential algorithms. One of its uses is to obtain upper bounds on the number of enumerated objects depending on the input graph. In [7] it is shown that the number of minimal dominating sets in an n-vertex graph is at most 1.7159^n. The *output-sensitive* approach measures the time complexity of an enumeration algorithm in the sum of the sizes of the input and the output. An algorithm whose running time is bounded by a polynomial depending on the sum of the sizes of the input and the output is called an *output-polynomial time* algorithm. In this paper we deal with the output-sensitive approach.

The existence of an output-polynomial time algorithm for the enumeration of minimal dominating sets of graphs is closely related to the well-known *Transversal problem* in hypergraphs. A *transversal* (or *hitting set*) in a hypergraph is a subset of its ground set that intersects every hyperedge. The *Transversal problem* asks for an output-polynomial time algorithm for the enumeration of all the (inclusion-wise) minimal transversals of hypergraphs. It is a long-standing open problem whether such an algorithm exists. The question is well-studied due to applications in several areas [5,6,8,13,15], and output-polynomial time algorithms are known for various restricted classes of hypergraphs (a summary of some known tractable cases is given in [10]. It is known that the set of minimal dominating sets of a graph is the same as the set of minimal transversals of its *closed neighbourhood hypergraph* [3]. Therefore, whenever the closed neighbourhood hypergraphs of a graph class is in one of the known tractable classes of hypergraphs, there exists an output-polynomial time algorithm for the enumeration of minimal dominating sets of graphs in this graph class. For example this is the case for degenerate graphs, line graphs, path-graphs, ... [10,12]. However, there are classes of closed neighbourhood hypergraphs for which no output-polynomial time algorithm of the transversal problem was known, *e.g.*, closed neighbourhood hypergraphs of split graphs for instance. Moreover, Kanté et al. [11] have shown that there exists an output-polynomial time algorithm for the enumeration of minimal transversals in hypergraphs if and only if there exists one for the enumeration of minimal dominating sets in graphs.

An enumeration algorithm is said to be of *linear delay* if it performs a polynomial time pre-processing in the size of the input and such that the delay between two consecutive outputs o_i and o_{i+1} is linear in the size of o_{i+1}. It is clear that linear delay enumeration algorithms are output-polynomial time algorithms. We give linear delay algorithms for the enumeration of minimal dominating sets in interval and permutation graphs, to achieve this we rely on the interval and the permutation model respectively. This improves considerably upon the known algorithms on interval graphs (the best known is designed for β-acyclic hypergraphs, and has delay between two consecutive outputs polynomial in the size of the input [5,6]). Moreover there are polynomial time counting algorithms to compute the number of minimal dominating sets of the input interval and permutation graph, respectively. We are not aware of such counting algorithms prior to our work. Linear delay is the best we can hope for whenever we want to list the elements of each minimal dominating set. Our techniques can be summarised as follows.

1. We first build in polynomial time a directed acyclic graph (DAG for short) and prove that some paths in this DAG correspond exactly to minimal dominating sets.
2. These paths can be counted in linear time (in the size of the DAG), and can be listed in linear delay with a classical *Depth First Search algorithm.*

Let us describe briefly the construction of the DAG in interval graphs. One first observes that any minimal dominating set in an interval graph is a collection of paths. Secondly, by using the interval model and by ordering the intervals (say their left endpoints from left to right), one can construct any minimal dominating set. In fact if you take a vertical line and those vertices X whose intervals are on the left of that vertical line, then for any minimal dominating set D either $D \cap X$ is a dominating set of the graph induced by X, or the vertex in D just on the right of the vertical line (following the ordering) should be adjacent to those vertices in X not dominated by $D \cap X$. We show, by moving in the right way such a vertical line, that we can construct any minimal dominating set by only keeping track of the last two chosen vertices. Following that, we can construct the DAG, the vertices of which will be those pairs (x, y) (with the left endpoint of x before the left endpoint of y) such that x and y can be both in a minimal dominating set, and the arcs are of the form $((x, y), (y, z))$ such that

– there is no vertex with its interval between the right endpoint of y and the left endpoint of z,
– $\{x, y, z\}$ can belong to a minimal dominating set.

We show that the minimal dominating sets of an interval graph are exactly those sets $\{x_{i_1}, \dots, x_{i_{k+1}}\}$ such that there exists a path $(v_1, \dots, v_k)$ with $v_j := (x_{i_j}, x_{i_{j+1}})$ with no intervals before the left endpoint of x_{i_1} and after the right endpoint of $x_{i_{k+1}}$.

For permutation graphs, the construction is obtained in a similar manner, but is more complicated. In fact each minimal dominating set can be still constructed from left to right (by ordering the bottom and top lines from left to right), but we need to keep track of the last three vertices, and following how their segments intersect, we need to keep one or two additional vertices. We postpone the details in Section 4.

Summary. In Section 2 we give some necessary definitions and we deal with interval graphs in Section 3. Permutation graphs are considered in Section 4. Some concluding remarks are given in Section 5.

2 Preliminaries

If A and B are two sets, $A \backslash B$ denotes the set $\{x \in A \mid x \notin B\}$. The power-set of a set V is denoted by 2^V. The size of a set A is denoted $|A|$.

We refer to [1] for graph terminology. The vertex set of a (directed) graph G is denoted by V_G and its edge set (or arc set) by E_G. We only deal with finite and simple (directed) graphs. We denote by n the size of the vertex set of a (directed) graph and by m the size of its edge (or arc) set. An arc from x to y in a directed graph is denoted by (x, y) and an edge between x and y in a graph is denoted by xy.

Let G be a graph. For a vertex x, we denote by $N_G(x)$ the set $\{y \in V_G \mid xy \in E_G\}$, and we let $N_G[x]$ be $N_G(x) \cup \{x\}$. For $X \subseteq V_G$, we write $N_G[X]$ and $N_G(X)$ for respectively $\bigcup_{x \in X} N_G[x]$ and $N_G[X] \setminus X$. We say that a vertex y is a *private neighbour with respect to* $D \subseteq V_G$ *of* x if $y \in N_G[x] \setminus N_G[D \setminus x]$. (When D is clear from the context, for convenience we will omit the expression "with respect to D".) Note that a private neighbour of a vertex $x \in D$ is either x itself, or a vertex in $V_G \setminus D$, but never a vertex $y \in D \setminus \{x\}$. The set of private neighbours of $x \in D$ is denoted by $P_D(x)$. A subset D of V_G is called an *irredundant set* if for all $x \in D$, we have $P_D(x) \neq \emptyset$. A subset D of V_G is called a *minimal dominating set* if it is an irredundant set, and each vertex in $V_G \setminus D$ has a neighbour in D.

A graph is an *interval graph* if it has an intersection model consisting of intervals on a straight line. A graph is a *permutation graph* if it has an intersection model consisting of straight lines between two parallels. We assume without loss of generality that any interval graph (or permutation graph) is given with its intersection model. Indeed, the recognition and a construction of an intersection model can be done in linear time for any interval graph (or permutation graph). See for instance [2] for interval graphs and [14] for permutation graphs.

Given a graph G and a subset $\mathcal{C}$ of 2^{V_G}, we say that an algorithm enumerates $\mathcal{C}$ with *linear delay* if, after a pre-processing that takes time $p(n+m)$ for some polynomial p, it outputs the elements of $\mathcal{C}$ without repetitions, the delay between two consecutive outputs o_i and o_{i+1} is bounded by $O(|o_{i+1}|)$. It is worth noticing that an algorithm which enumerates a subset $\mathcal{C}$ of 2^{V_G} in linear delay outputs the set $\mathcal{C}$ in time $O\left(p(n+m) + \sum_{C \in \mathcal{C}} |C|\right)$ where p is the polynomial bounding the pre-processing time.

We finish these preliminaries with the following folklore theorem on the enumeration and the counting of maximal paths in directed acyclic graphs.

Theorem 1 (folklore). *Given a directed acyclic graph D and two disjoint subsets S and P of vertices of D, the enumeration of paths from vertices in S to vertices in P can be done in linear delay. Moreover, counting these paths can be done in linear time in the size of D.*

3 Interval Graphs

We may suppose without loss of generality that in an intersection model of an interval graph all endpoints are pairwise distinct. For an interval graph G, let us denote its interval model by I_G, and for each vertex x of G let $I_G(x)$ be the interval in I_G associated with x. We number the endpoints of the intervals from left to right and we denote by $s(x)$ and $e(x)$ the left and right endpoint of $I_G(x)$ respectively. We can therefore assume that $I_G(x) := [s(x), e(x)]$ and will be viewed as the set of points on the line between $s(x)$ and $e(x)$.

We linearly order the vertices of an interval graph G with the linear order $\preceq$ such that $x \preceq y$ whenever $s(x) \leq s(y)$. We can therefore consider that the vertices of G are enumerated as $x_1, x_2, \ldots, x_n$ with $x_i \preceq x_j$ whenever $i \leq j$. We will, in the sequel, consider any subset D of V_G as linearly ordered by $\preceq$ and when we write $\{x_{i_1}, \ldots, x_{i_k}\}$, then we consider $x_{i_j} \preceq x_{i_\ell}$ whenever $j \leq \ell$. The proof of the following lemma is straightforward.

Lemma 2. *Let D be an irredundant set of an interval graph G. Then for all distinct vertices x and y in D, the sets $I_G(x) \setminus I_G(y)$ and $I_G(y) \setminus I_G(x)$ are non empty.*

The following is an easy corollary of Lemma 2.

Corollary 3. *Let D be a minimal dominating set of an interval graph G. Then for all distinct vertices x and y in D, we have $e(x) < e(y)$ whenever $x \prec y$.*

For $x \in V_G$, we let $NC(x)$ be the set $\{y \in V_G \mid s(y) > e(x)\}$, and $nc^s(x)$ and $nc^e(x)$ be respectively $\min\{s(y) \mid y \in NC(x)\}$ and $\min\{e(y) \mid y \in NC(x)\}$. Notice that if y is such that $s(y) = nc^s(x)$, then we do not have necessarily $e(y) = nc^e(x)$, and vice-versa. For $D := \{x_{i_1}, \ldots, x_{i_k}\}$ a subset of the vertex set of an interval graph G and $j \leq k$, we denote by D_j the subset $\{x_{i_1}, \ldots, x_{i_j}\}$ of D, and we let $p_D(x_{i_j}) := \min\{e(y) \mid y \in P_{D_j}(x_{i_j})\}$.

Lemma 4. *Let $D := \{x_{i_1}, \ldots, x_{i_k}\}$ be an irredundant set of an interval graph G. For $j \leq k$, let x_{p_j} be such that $e(x_{p_j}) = p_D(x_{i_j})$. Then $x_{p_j} \in P_D(x_{i_j})$.*

The following characterises minimal dominating sets in interval graphs, and will be the core of our algorithm.

Proposition 5. *A subset $D := \{x_{i_1}, \ldots, x_{i_k}\}$ of an interval graph G is a minimal dominating set if and only if the following conditions hold.*

1. *For all $\ell \leq n$, we have $s(x_{i_1}) \leq e(x_\ell)$.*
2. *For all $\ell \leq n$, we have $e(x_{i_k}) \geq s(x_\ell)$.*
3. *For all $j \leq k$, we have $j = k$ if $NC(x_{i_j}) = \emptyset$, otherwise we have $e(x_{i_{j+1}}) \geq nc^s(x_{i_j})$.*
4. *For all $1 \leq j < k$, we have $p_D(x_{i_j}) < s(x_{i_{j+1}}) \leq nc^e(x_{i_j})$.*

The following tells us how to compute the private neighbour of a vertex.

Proposition 6. *Let $D := \{x_{i_1}, \ldots, x_{i_k}\}$ be an irredundant set. Then for all $x \in N_G[x_{i_k}]$, we have $x \in P_D(x_{i_k})$ if and only if $e(x_{i_{k-1}}) < s(x)$.*

In order to enumerate in linear delay, and count in polynomial time, the set of minimal dominating sets of an interval graph G, we associate with it a DAG, denoted by $Dag_I(G)$, where the paths from a subset of the sources to a subset of the sinks correspond exactly to minimal dominating sets of G. Let G be an interval graph. The graph $Dag_I(G)$ has vertex set the pairs (x_i, x_j) such that

$$\begin{aligned} &(V.1) \qquad && x_i \preceq x_j, \\ &(V.2) \qquad && p_{\{x_i\}}(x_i) < s(x_j) \leq nc^e(x_i), \\ &(V.3) \qquad && NC(x_i) \neq \emptyset \text{ and } e(x_j) \geq nc^s(x_i), \end{aligned}$$

and it has as arc set the set of pairs $((x_i, x_j), (x_j, x_k))$ such that

$$(E.1) \qquad p_{\{x_i,x_j\}}(x_j) < s(x_k) \leq nc^e(x_j).$$

A vertex (x_i, x_j) of $Dag_I(G)$ is called an *initial vertex* if $s(x_i) \leq e(x_\ell)$ for all $1 \leq \ell \leq n$. A vertex (x_i, x_j) of $Dag_I(G)$ is called a *final vertex* if $e(x_j) \geq s(x_\ell)$ for all $1 \leq \ell \leq n$, and then $NC(x_j) = \emptyset$.

Lemma 7. *For every interval graph G, we have the following.*

1. *$Dag_I(G)$ is a DAG.*
2. *If a vertex (x_i, x_j) of $Dag_I(G)$ is an initial vertex (resp. a final vertex), then it is a source (resp. a sink) of $Dag_I(G)$.*
3. *$Dag_I(G)$ can be constructed in time $O(n^3)$.*

Proposition 8. *Let G be an interval graph and let v_1 and v_k be respectively an initial vertex and a final vertex of $Dag_I(G)$. Then $(v_1, v_2, \ldots, v_k)$ is a path of $Dag_I(G)$ if and only if $\{x_{i_1}, \ldots, x_{i_{k+1}}\}$ is a minimal dominating set of G of size greater than or equal to 2 with $v_j := (x_{i_j}, x_{i_{j+1}})$.*

We can now state the main theorem of the section.

Theorem 9. *Let G be an interval graph. Then, after a pre-processing in time $O(n^3)$, one can enumerate in linear delay the minimal dominating sets of G. One can moreover count them in time $O(n^3)$.*

Proof. By Lemma 7 the DAG $Dag_I(G)$ can be constructed in time $O(n^3)$. By Proposition 8, there is a bijection between paths from initial vertices to final vertices in $Dag_I(G)$ and minimal dominating sets of G of size at least 2.

It remains to deal now with minimal dominating sets of size 1. For each x, we can determine in time $O(n)$ if $\{x\}$ is a minimal dominating set. So, let $S := \{x \in V_G \mid \{x\}$ is a minimal dominating set of $G\}$. The set S can be constructed at the same time as $Dag_I(G)$. We let G' be the DAG obtained from $Dag_I(G)$ by adding new vertices v_x to $Dag_I(G)$, with in-degree and out-degree 0, for each $x \in S$. Therefore, each such new vertex v_x is a source and a sink at the same time. We define the following subsets of $V_{G'}$.

$$S := \{v_x \mid x \in S\} \cup \{v \in V_{Dag_I(G)} \mid v \text{ is an initial vertex}\},$$
$$T := \{v_x \mid x \in S\} \cup \{v \in V_{Dag_I(G)} \mid v \text{ is a final vertex}\}.$$

It is clear now that paths from S to T in G' are in bijection with all the minimal dominating sets of G. Since, paths from S to T in DAGs can be listed in linear delay, and be counted in linear time (see Theorem 1), we are done. □

4 Permutation Graphs

For a permutation graph G let us denote its permutation model by L_G, and for each vertex x of G let $L_G(x)$ be the segment in L_G corresponding to x. We number the endpoints of segments from left to right and we denote by $b(x)$ and $t(x)$ the endpoints of $L_G(x)$ on the bottom line and top line respectively. All endpoints are assumed to be different without loss of generality. We order the vertices of G by their bottom line endpoints, and then the vertices of G are assumed to be enumerated as $x_1, \ldots, x_n$ where $b(x_i) \leq b(x_j)$ whenever $i \leq j$. As in the interval case, we will also consider any subset D of V_G as linearly ordered, and when we write $\{x_{i_1}, \ldots, x_{i_k}\}$, then we consider $i_1 < \ldots < i_k$ and hence $b(x_{i_1}) < \cdots < b(x_{i_k})$. For two vertices x and y of G, we say that $L_G(x) < L_G(y)$ whenever $b(x) < b(y)$ and $t(x) < t(y)$.

Lemma 10. *Let D be an irredundant set of a permutation graph G. Then $G[D]$ contains neither triangles nor claws. Therefore, for each $x \in D$, $d_{G[D]}(x) \leq 2$.*

For a subset $D := \{x_{i_1}, \ldots, x_{i_{k+1}}\}$ of G, if $k \geq 4$, we let x_D be the vertex $x_{i_r} \in D$ such that $t(x_{i_r}) := \max\{t(x_{i_\ell}) \mid x_{i_\ell} \in D \text{ and } \ell < k-2\}$, i.e. x_D is the top right most vertex of D when we remove the four greatest vertices of D. If $k \geq 3$, we let $A(D)$ be the set formed by the four greatest vertices of D and x_D if it exists, i.e. for $k \geq 3$,

$$A(D) := \begin{cases} D & \text{if } k = 3, \\ \{x_D, x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}, x_{i_{k+1}}\} & \text{if } k \geq 4. \end{cases}$$

Lemma 11. *Let $D' := \{x_{i_1}, x_{i_2}, ..., x_{i_{k+1}}\}$ with $k \geq 4$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$ and $A(D')$ are irredundant sets of G. Then for all $l \leq i_{k-3}$, $N_G[x_{i_{k+1}}] \cap P_D(x_{i_l}) = \emptyset$.*

In the next lemmas we show how to construct irredundant sets of a permutation graph G from left to right. Indeed, we characterise exactly the situations where an irredundant set $D := \{x_{i_1}, \ldots, x_{i_k}\}$ can be extended to an irredundant set $D' := D \cup \{x_{i_{k+1}}\}$, and we show that for deciding the extension we only need to know $x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}$, and following the intersections of $L_G(x_{i_{k-2}}), L_G(x_{i_{k-1}})$ and $L_G(x_{i_k})$ we need to know also either $x_{D'}$ or the vertex x_s such that $t(x_s) := \min\{t(z) \mid z \in P_D(y)\}$ with y such that $t(y) = \min\{t(x_{i_{k-2}}), t(x_{i_{k-1}}), t(x_{i_k})\}$, or both. The cases summarising the intersections of $L_G(x_{i_{k-2}}), L_G(x_{i_{k-1}})$ and $L_G(x_{i_k})$ are depicted in Fig. 1.

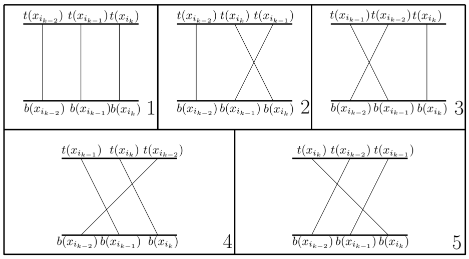

$t(x_{i_{k-2}})$ $t(x_{i_{k-1}})$ $t(x_{i_k})$
$b(x_{i_{k-2}})$ $b(x_{i_{k-1}})$ $b(x_{i_k})$
1
$t(x_{i_{k-2}})$ $t(x_{i_k})$ $t(x_{i_{k-1}})$
$b(x_{i_{k-2}})$ $b(x_{i_{k-1}})$ $b(x_{i_k})$
2
$t(x_{i_{k-1}})$ $t(x_{i_{k-2}})$ $t(x_{i_k})$
$b(x_{i_{k-2}})$ $b(x_{i_{k-1}})$ $b(x_{i_k})$
3
$t(x_{i_{k-1}})$ $t(x_{i_k})$ $t(x_{i_{k-2}})$
$b(x_{i_{k-2}})$ $b(x_{i_{k-1}})$ $b(x_{i_k})$
4
$t(x_{i_k})$ $t(x_{i_{k-2}})$ $t(x_{i_{k-1}})$
$b(x_{i_{k-2}})$ $b(x_{i_{k-1}})$ $b(x_{i_k})$
5

Fig. 1. Different cases following the intersections of $L_G(x_{i_{k-2}}), L_G(x_{i_{k-1}})$ and $L_G(x_{i_k})$

Lemma 12 (Case 1). *Let $D' := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ with $k \geq 3$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set of G and $\{x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}\}$ corresponds to Case (1) of Fig. 1. Then D' is an irredundant set of G if and only if*

1. *$A(D')$ is an irredundant set of G,*
2. *$t(x_{i_{k+1}}) > \min\{t(y) \mid y \in P_D(x_{i_{k-2}})\}$.*

Lemma 13 (Case 2). *Let $D' := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ with $k \geq 3$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set of G and $\{x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}\}$ corresponds to Case (2) of Fig. 1. Then D' is an irredundant set of G if and only if $A(D')$ is an irredundant set of G.*

Lemma 14 (Case 3). *Let $D' := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ with $k \geq 3$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set of G and $\{x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}\}$ corresponds to Case (3) of Fig. 1. Then D' is an irredundant set of G if and only if*

1. *$A(D')$ is an irredundant set of G,*
2. *$t(x_{i_{k+1}}) > \min\{t(y) \mid y \in P_D(x_{i_{k-1}})\}$.*

Lemma 15 (Case 4). *Let $D' := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ with $k \geq 3$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set of G and $\{x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}\}$ corresponds to Case (4) of Fig. 1. Then D' is an irredundant set of G if and only if $A(D')$ is an irredundant set of G.*

Lemma 16 (Case 5). *Let $D' := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ with $k \geq 3$ be a subset of the vertex set of a permutation graph G such that $D := \{x_{i_1}, \ldots, x_{i_k}\}$ is an irredundant set of G and $\{x_{i_{k-2}}, x_{i_{k-1}}, x_{i_k}\}$ corresponds to Case (5) of Fig. 1. Then D' is an irredundant set of G if and only if*

1. *$A(D') \setminus x_{D'}$ is an irredundant set of G,*
2. *$t(x_{i_{k+1}}) > \min\{t(v) \mid v \in P_D(x_{i_k})\}$.*

The next proposition shows that minimal dominating sets are exactly those irredundant sets $\{x_{i_1}, \ldots, x_{i_k}\}$ with no segments smaller than $L_G(x_{i_1})$ (greater than $L_G(x_{i_k})$), and for each $2 \leq l < k$, all vertices y with $L_G(x_{i_{l-2}}) < L_G(y)$ are in $N_G[D_{l+1}]$.

Proposition 17. *Let $D := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ be an irredundant set of a permutation graph G. Then D is a minimal dominating set of G if and only if the following conditions are fulfilled*

1. *for each l, $t(x_l) \geq \min(t(x_{i_1}), t(x_{i_2}), t(x_{i_3}))$ or $b(x_l) \geq b(x_1)$,*
2. *$\{y \mid L_G(x_{i_{k-2}}) < L_G(y)$ and $y \notin N_G[D]\} = \emptyset$,*
3. *for all $2 \leq l < k$, if $L_G(x_{i_l})$ does not intersect $L_G(x_{i_{l-1}})$ then $\{y \mid L_G(x_{i_{l-2}}) < L_G(y) < L_G(x_{i_{l-1}})$ and $y \notin N_G[D_{l+1}]\} = \emptyset$. Furthermore, $\{y \mid L_G(x_{i_{l-1}}) < L_G(y)$, $t(y) < t(x_{i_{l+1}})$, and $b(y) < b(x_{i_l})$ and $y \notin N_G[D_{l+1}]\} = \emptyset$ if $L_G(x_{i_{l+1}})$ intersects $L_G(x_{i_l})$.*

Proposition 18. *Let $D := \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_{k+1}}\}$ be a subset of the vertex set of a permutation graph G and let $x \in \{y \mid L_G(x_{i_{k-2}}) < L_G(y)\}$. Then $x \in N_G[D]$ if and only if $x \in N_G[A(D)]$.*

Let G be a permutation graph and let $\perp$ be a non vertex of G. Thanks to Lemmas 12-16, and Propositions 17 and 18, the DAG to which some paths correspond to the minimal dominating sets of G will have as vertices those quintuple $(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5})$ in $(V_G \cup \perp)^5$ that correspond to Cases (1)-(5) of Fig. 1, and the arcs describe the construction of minimal dominating sets from left to right (and thanks to Proposition 18 the arcs can be constructed in polynomial time).

We denote by Initial(G) those quintuples $(x_i, x_j, x_k, \perp, x_s) \in (V_G \cup \perp)^5$ such that $A := \{x_i, x_j, x_k\}$ is an irredundant set of G, and one of the following conditions hold

(I.1) $t(x_s) := \min\{t(y) \mid y \in P_A(x_i)\}$ if A corresponds to Case (1) of Fig. 1 and $\{y \mid L_G(y) < L_G(x_i)\} = \emptyset$.
(I.2) $x_s = \perp$ if A corresponds to Case (2) of Fig. 1 and $\{y \mid t(y) < t(x_k),\ b(y) < b(x_j),\ \mathit{and}\ y \notin N_G[A]\} = \emptyset$.
(I.3) $t(x_s) := \min\{t(y) \mid y \in P_A(x_j)\}$ if A corresponds to Case (3) of Fig. 1 and $\{y \mid t(y) < t(x_j)\ \mathit{and}\ b(y) < b(x_i)\} = \emptyset$.
(I.4) $x_s = \perp$ if A corresponds to Case (4) of Fig. 1 and $\{y \mid t(y) < t(x_j),\ b(y) < b(x_i),\ \mathit{and}\ y \notin N_G[A]\} = \emptyset$.
(I.5) $t(x_s) := \min\{t(y) \mid y \in P_A(x_k)\}$ if A corresponds to Case (5) of Fig. 1 and $\{y \mid t(y) < t(x_k)\ \mathit{and}\ b(y) < b(x_i)\} = \emptyset$.

We denote by Regular(G) those quintuples $(x_i, x_j, x_k, x_r, x_s) \in (V_G \cup \perp)^5$ such that $A := \{x_i, x_j, x_k\}$ is an irredundant set of G, and one of the following conditions hold

(R.1) $r < i$ and $x_s = \perp$ if A corresponds to Case (2) or (4) of Fig. 1.
(R.2) $r < i$ and $x_s \in N_G[x_i] \setminus N_G[\{x_r, x_j, x_k\}]$ if A corresponds to Case (1) of Fig. 1.
(R.3) $r < i$ and $x_s \in N_G[x_j] \setminus N_G[\{x_r, x_i, x_k\}]$ if A corresponds to Case (3) of Fig. 1.
(R.4) $x_r = \perp$ and $x_s \in N_G[x_k] \setminus N_G[\{x_r, x_i, x_j\}]$ if A corresponds to Case (5) of Fig. 1.

For $v := (x_i, x_j, x_k, x_r, x_s)$ in Initial$(G) \cup$ Regular(G) we let

$$A(v) := \begin{cases} \{x_r, x_i, x_j, x_k\} & \text{if } x_r \neq \perp, \\ \{x_i, x_j, x_k\} & \text{otherwise.} \end{cases}$$

We let $Dag_P(G)$ be the DAG with vertex set Initial$(G) \cup$ Regular(G) and such that there is an arc (v_1, v_2) with $v_1 := (x_i, x_j, x_k, x_r, x_s)$ and $v_2 := (x_j, x_k, x_l, x_{r'}, x_{s'})$ if the following conditions are satisfied.

(A.1) If $x_{r'} \neq \perp$, then $t(x_{r'}) = \max(t(x_r), t(x_i))$ if $x_r \neq \perp$, otherwise $t(x_{r'}) = x_i$.
(A.2) $A := A(v_1) \cup \{x_l\}$ is an irredundant set of G.
(A.3) $t(x_l) > t(x_s)$ if $x_s \neq \perp$.

(A.4) If $L_G(x_k)$ does not intersect $L_G(x_j)$ then $\{y \mid L_G(x_i) < L_G(y) < L_G(x_j)$ *and* $y \notin N_G[A]\} = \emptyset$. Furthermore, $\{y \mid L_G(x_j) < L_G(y),\ t(y) < t(x_l)$, *and* $b(y) < b(x_k)$ *and* $y \notin N_G[A]\} = \emptyset$ if $L_G(x_l)$ intersects $L_G(x_k)$.
(A.5) If $x_{s'} \neq \perp$ then
(A.5.1) $x_s = x_{s'}$ if $x_s \neq \perp$ and $\min\{t(x_i), t(x_j), t(x_k)\} = \min\{t(x_j), t(x_k), t(x_l)\}$.
(A.5.2) $t(x_{s'}) = \min\{t(x) \mid x \in P_A(y)\}$ where y is the vertex such that $t(y) = \min\{t(x_j), t(x_k), t(x_l)\}$ otherwise.

A vertex v of $Dag_P(G)$ is called an *initial vertex* if it belongs to Initial(G) and it is called a *final vertex* if $\{y \mid L_G(x_i) < L_G(y)$ *and* $y \notin N_G[A(v)]\} = \emptyset$. The set of final vertices is denoted by Final(G).

Proposition 19. *Let G a be permutation graph. A subset $D := \{x_{i_1}, x_{i_2}, ..., x_{i_{k+2}}\}$ of V_G is a minimal dominating set of G of size greater than or equal to three, if and only if there exists a path $(v_1, ..., v_k)$ of $Dag_P(G)$ where $v_j := (x_{i_j}, x_{i_{j+1}}, x_{i_{j+2}}, x_{r_j}, x_{s_j})$, and $v_1 \in$ Initial(G) and $v_k \in$ Final(G).*

Theorem 20. *Let G be a permutation graph. Then, after a pre-processing in time $O(n^8)$, one can enumerate in linear delay the minimal dominating sets of G. One can moreover count them in time $O(n^8)$.*

5 Conclusion

If we want to list a subset $\mathcal{C} \subseteq 2^{V_G}$ of a graph G by outputting each element of each $C \in \mathcal{C}$, then the size of $\mathcal{C}$ defined as $\sum_{C \in \mathcal{C}} |C|$ is a lower bound. We have proposed linear delay algorithms for the enumeration of minimal dominating sets in interval and permutation graphs the running times of which match the above lower bound. Our techniques allow also a polynomial time algorithm (in the sizes of the graphs) for counting minimal dominating sets. The results presented here may be extended to *trapezoid graphs*, but the proofs are more tricky. It is not known whether one can enumerate minimal dominating sets in *circle graphs*. Can we adapt some of our techniques to them?

Acknowledgement. A. Mary and L. Nourine are partially supported by the ANR (french National Research Agency) project DAG (ANR-09-DEFIS, 2009-2012). M.M. Kanté and V. Limouzy are supported by the ANR junior project DORSO (2011-2015).

References

1. Bondy, A., Murty, U.S.R.: Graph Theory. Graduate Texts in Mathematics. Springer (2008)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. J. Comput. Syst. Sci. 13(3), 335–379 (1976)

3. Brandstädt, A., Bang Le, V., Spinrad, J.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications, vol. 3. Society for Industrial and Applied Mathematics (1999)
4. Chao, H.S., Hsu, F.-R., Lee, R.C.T.: An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs. Discrete Applied Mathematics 102(3), 159–173 (2000)
5. Eiter, T., Gottlob, G.: Identifying the minimal transversals of a hypergraph and related problems. SIAM J. Comput. 24(6), 1278–1304 (1995)
6. Eiter, T., Gottlob, G., Makino, K.: New results on monotone dualization and generating hypergraph transversals. SIAM J. Comput. 32(2), 514–537 (2003)
7. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. Algorithms 5(1), 1–17 (2008)
8. Gunopulos, D., Khardon, R., Mannila, H., Toivonen, H.: Data mining, hypergraph transversals, and machine learning. In: PODS, pp. 209–216 (1997)
9. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Pure and Applied Mathematics, vol. 208. Marcel Dekker (1998)
10. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of minimal dominating sets and variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011)
11. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: On the enumeration of minimal dominating sets and related notions. In: Submitted (2012)
12. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 289–298. Springer, Heidelberg (2012)
13. Khachiyan, L., Boros, E., Elbassioni, K.M., Gurvich, V.: An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. Discrete Applied Mathematics 154(16), 2350–2372 (2006)
14. McConnell, R.M., Spinrad, J.: Modular decomposition and transitive orientation. Discrete Mathematics 201(1-3), 189–241 (1999)
15. Nourine, L., Petit, J.-M.: Extending set-based dualization: Application to pattern mining. In: ECAI. IOS Press, Montpellier (2012)
16. Chang, M.S.: Efficient algorithms for the domination problems on interval and circular-arc graphs. SIAM J. Comput. 27, 1671–1694 (1998)

Testing Mutual Duality of Planar Graphs*

Patrizio Angelini[1], Thomas Bläsius[2], and Ignaz Rutter[2]

[1] Università Roma Tre
angelini@dia.uniroma3.it
[2] Karlsruhe Institute of Technology (KIT)
{blaesius,rutter}@kit.edu

Abstract. We introduce and study the problem MUTUAL PLANAR DUALITY, which asks for planar graphs G_1 and G_2 whether G_1 can be embedded such that its dual is isomorphic to G_2. We show NP-completeness for general graphs and give a linear-time algorithm for biconnected graphs.

We consider the *common dual relation* $\sim$, where $G_1 \sim G_2$ if and only they admit embeddings that result in the same dual graph. We show that $\sim$ is an equivalence relation on the set of biconnected graphs and devise a succinct, SPQR-tree-like representation of its equivalence classes. To solve MUTUAL PLANAR DUALITY for biconnected graphs, we show how to do isomorphism testing for two such representations in linear time.

A special case of MUTUAL PLANAR DUALITY is testing whether a graph is self-dual. Our algorithm can handle the case of biconnected graphs in linear time and our NP-hardness proof extends to self-duality and also to map self-duality testing (which additionally requires to preserve the embedding).

1 Introduction

Let G be a planar graph with embedding $\mathcal{G}$ and let F be the set of faces of $\mathcal{G}$. The *dual* of G with respect to $\mathcal{G}$ is the graph $G^\star = (F, E^\star)$ with $E^\star = \{e^\star \mid e \in E\}$. The *dual edge* $e^\star$ of e connects the two faces incident to e in $\mathcal{G}$. Thus, $G^\star$ models the adjacencies of faces of G with respect to $\mathcal{G}$. We consider the problem MUTUAL PLANAR DUALITY. Given two planar graphs G_1 and G_2, is there an embedding $\mathcal{G}_1$ of G_1 such that the dual $G_1^\star$ of G_1 with respect to $\mathcal{G}_1$ is isomorphic to G_2? All graphs we consider are implicitly allowed to have multiple edges and loops. If G_1 is triconnected, it has a fixed planar embedding [13] and thus MUTUAL PLANAR DUALITY reduces to testing graph isomorphism for planar graphs, which is linear-time solvable due to Hopcroft and Wong [8]. Observe that bi- and triconnectivity of a planar graph is invariant under dualization [12]. For non-triconnected planar graphs MUTUAL PLANAR DUALITY is more complicated, since changing the embedding of G_1 influences the structure of its dual graph. In fact, we show that MUTUAL PLANAR DUALITY is NP-complete in general.

On the other hand, for biconnected planar graphs we provide a linear-time algorithm solving MUTUAL PLANAR DUALITY that is based on the definition of a new data structure that we call *dual SPQR-tree* in analogy with the SPQR-tree [5]. The dual SPQR-trees, together with a newly-defined set of operations, allows to succinctly represents and

* Partially supported by ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations". This work began during a visit of Angelini at Karlsruhe Institute of Technology.

efficiently handle all the dual graphs of a biconnected planar graph. This data structure has an interesting implication on the structure of the dual graphs of a biconnected planar graph. Namely, consider the *common dual relation* $\sim$, where $G_1 \sim G_2$ if and only if they have a common dual graph. We show that $\sim$ is not transitive on the set of connected planar graphs. However, it follows from the dual SPQR-tree that $\sim$ is an equivalence relation on the set of biconnected planar graphs. In particular, the graphs represented by a dual SPQR-tree form an equivalence class. Thus, testing MUTUAL PLANAR DUALITY reduces to testing whether two dual SPQR-trees represent the same equivalence class. It is not hard to see that two biconnected graphs are related via the common dual relation if and only if they have the same graphic matroid (which again does not hold for general planar graphs). With this insight, one can use the one-to-many reduction from graphic matroid isomorphism testing to graph isomorphism testing by Rao and Sarma [9] to solve MUTUAL PLANAR DUALITY for biconnected planar graphs in polynomial time. We give a one-to-one reduction leading to a linear-time algorithm.

We note that the common-dual relation is closely related to 2-isomorphisms, studied by Whitney [14]. Two graphs are 2-isomorphic if and only if their cycle matroids are isomorphic. On biconnected graphs, the notions coincide, and our algorithm implies a linear-time isomorphism testing algorithm for graphic matroids of planar graphs.

We believe that the new data structure of dual SPQR-trees can be used to efficiently solve other related problems. In many applications it is desirable to find an embedding of a given planar graph that optimizes certain criteria, which can often be naturally described in terms of the dual graph with respect to the chosen embedding. For example, Bienstock and Monma [4], and Angelini et al. [1] seek an embedding of a planar graph minimizing the largest distance of internal faces to the external face. In terms of the dual graph this corresponds to minimizing the largest distance of a vertex to all other vertices. For problems of this kind it can be useful to work directly with a representation of all dual graphs, instead of taking the detour over a representation of all planar embeddings.

We finally remark that MUTUAL PLANAR DUALITY is a generalization of the self-duality of planar graphs [10]. A graph G is *graph self-dual* if it admits an embedding such that its dual $G^\star$ is isomorphic to G. We call the corresponding decision problem GRAPH SELF-DUALITY. A stronger form of self-duality is defined as follows. A graph G is *map self-dual* [11] if and only if G has an embedding $\mathcal{G}$ such that there exists an isomorphism from G to its dual graph $G^\star$ that preserves embedding $\mathcal{G}$. The corresponding decision problem is called MAP SELF-DUALITY. Since triconnected planar graphs have a unique planar embedding, GRAPH SELF-DUALITY and MAP SELF-DUALITY are equivalent for them. Servatius and Servatius [11] show the existence of biconnected planar graphs that are graph self-dual but not map self-dual. Servatius and Christopher [10] show how to construct self-dual graphs from given planar graphs. Archdeacon and Richter [3] give a set of constructions for triconnected self-dual graphs and show that every such graph can be constructed in this way. To the best of our knowledge the computational complexity of testing MAP or GRAPH SELF-DUALITY is open. Since GRAPH SELF-DUALITY is a special case of MUTUAL PLANAR DUALITY, our algorithm can be used to solve GRAPH SELF-DUALITY in linear time when G is biconnected. Moreover, our NP-hardness proof for general instances extends to MAP and GRAPH SELF-DUALITY.

Outline. In Section 2 we show that MUTUAL PLANAR DUALITY is NP-complete, even if both input graphs are required to be simple. The proof can be extended to show that MAP SELF-DUALITY and GRAPH SELF-DUALITY are NP-complete in general. To solve MUTUAL PLANAR DUALITY efficiently for biconnected graphs, we first describe decomposition trees as a generalization of SPQR-trees in Section 3. In Section 4 we describe the dual SPQR-tree and show that it succinctly represents all dual graphs of a biconnected planar graph. We consider the common dual relation in Section 5 and show that $\sim$ is not transitive on the set of connected planar graphs. On the other hand, we show that it follows from the dual SPQR-tree that $\sim$ is an equivalence relation on the set of biconnected planar graphs. This implies that solving MUTUAL PLANAR DUALITY is equivalent to testing whether two dual SPQR-trees represent the same equivalence class. In Section 6 we show that this reduces to testing graph isomorphism of two planar graphs, which leads to a linear-time algorithm for MUTUAL PLANAR DUALITY, including GRAPH SELF-DUALITY as a special case. Omitted proofs can be found in the full version of this paper[2].

2 Complexity

In this section we first show that MUTUAL PLANAR DUALITY is NP-complete by a reduction from 3-PARTITION. Then we show that the resulting instances of MUTUAL PLANAR DUALITY can be further reduced to equivalent instances of MAP and GRAPH SELF-DUALITY. An instance (A, B) of 3-PARTITION consists of a positive integer B and a set $A = \{a_1, \dots, a_{3m}\}$ of $3m$ integers with $B/4 < a_i < B/2$ for $i = 1, \dots, 3m$. The question is whether A admits a partition $\mathcal{A}$ into a set of triplets such that for each triplet $X \in \mathcal{A}$ we have $\sum_{x \in X} x = B$. The problem 3-PARTITION is strongly NP-hard [6], i.e., it remains NP-hard even if B is bounded by a polynomial in m.

Theorem 1. MUTUAL PLANAR DUALITY *is NP-complete, even for simple graphs.*

Proof. Clearly, MUTUAL PLANAR DUALITY is in NP, as we can guess an embedding for graph G_1 and then check in polynomial time whether $G_1^\star$ is isomorphic to G_2.

To show NP-hardness we give a reduction from 3-PARTITION. Our construction first contains loops, later we show how to get rid of them. Let (A, B) be an instance of 3-PARTITION with $|A| = 3m$. The graph G_1 contains a wheel of size m, i.e., a cycle $v_1, \dots, v_m$ together with a *center* u connected to each v_i. Additionally, for each element $a_i \in A$ we create a star T_i with $a_i - 1$ leaves and connect its center to u; see Figure 1(a). The graph G_2 is a wheel of size m along with B loops at every vertex except for the center; see Figure 1(b). We claim that G_1 and G_2 form a YES-instance of MUTUAL PLANAR DUALITY if and only if (A, B) is a YES-instance of 3-PARTITION.

Suppose that there exists a partition $\mathcal{A}$ of A. The embedding of the wheel in G_1 is fixed and it has exactly m faces incident to the center u. The remaining degree of freedom is to decide the embedding of the trees T_i into these m faces. For each triplet $X = \{a_i, a_j, a_k\} \in \mathcal{A}$ we pick a distinct such face and embed T_i, T_j and T_k into it. Call the resulting embedding $\mathcal{G}_1$ and consider the dual $G_1^\star$ with respect to $\mathcal{G}_1$. The wheel of G_1 determines a wheel of size m in $G_1^\star$. Consider a tree T_i that is embedded in a face f. Since T_i contains a_i bridges, which are all embedded in f, the corresponding

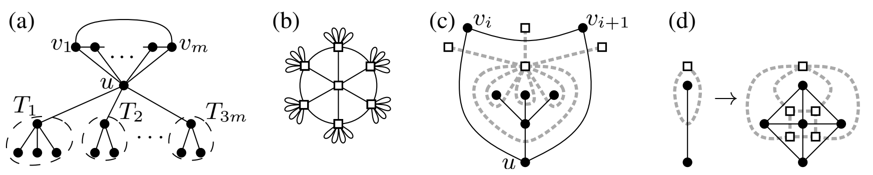


Fig. 1. The graphs G_1 (a) and G_2 (b) of the reduction from 3-PARTITION. (c) Embedding a tree T_i inside a face f creates a_i loops at the corresponding dual vertex. (d) Bridges and corresponding loops can be replaced by small graphs.

vertex of $G_1^\star$ has a_i loops; see Figure 1(c). Due to the construction each face contains exactly three trees with a total of B bridges. Thus, $G_1^\star$ is isomorphic to G_2.

Conversely, assume to have embeddings G_1 and G_2 such that the dual $G_1^\star$ of G_1 is isomorphic to G_2. Again, the wheel in G_1 forms m faces incident to u, and since $G_1^\star$ is isomorphic to G_2, the trees must be embedded such that each face contains exactly B bridges. Since embedding T_i inside a face f places a_i bridges into f and since $B/4 < a_i < B/2$, each face contains exactly three trees. Thus, the set of triplets determined by trees that are embedded in the same face form a solution to 3-PARTITION.

Clearly, the transformation can be computed in polynomial time, and thus MUTUAL PLANAR DUALITY is NP-hard. Moreover, the graph G_2 can be made simple (G_1 is already simple) by replacing each bridge in G_1 and each loop in G_2 with a 4-wheel as shown in Figure 1(d). The resulting graphs G_1' and G_2' are obviously dual to each other if and only if G_1 and G_2 are dual to each other. Moreover, G_1' and G_2' are simple. □

In the following we show how the above construction can be used to show NP-completeness for MAP and GRAPH SELF-DUALITY. To this end, we use the *adhesion* operation introduced by Servatius and Christopher [10]. Let v be a vertex of G incident to a face f. Then the adhesion of G and its dual $G^\star$ (with respect to v and f) is obtained by identifying v in G and $f^\star$ in $G^\star$. Servatius and Christopher [10] show that the adhesion of a plane graph and its dual is graph self-dual. Moreover, although not explicitly mentioned, they show that this adhesion is even map self-dual. To show the following theorem we essentially transform the instance of MUTUAL PLANAR DUALITY consisting of the two graphs G_1 and G_2 described in the proof of Theorem 1 into an equivalent instance of MAP and GRAPH SELF-DUALITY by forming the adhesion of G_1 and G_2.

Theorem 2. GRAPH SELF-DUALITY *and* MAP SELF-DUALITY *are NP-complete.*

Sketch of Proof. Both problems are in NP. Let G_1 and G_2 form an instance of MUTUAL PLANAR DUALITY obtained from an instance of 3-PARTITION as in the proof of Theorem 1. Let G be the graph obtained by identifying a vertex that is not the center of the wheel in G_2 with the vertex u in G_1. The theorem is implied by the following claims.
Claim 1. If G is a YES-instance of MAP SELF-DUALITY, it is a YES-instance of GRAPH SELF-DUALITY.
Claim 2. If G_1 and G_2 form a YES-instance of MUTUAL PLANAR DUALITY, then G is a YES-instance of MAP SELF-DUALITY.
Claim 3. If G is a YES-instance of GRAPH SELF-DUALITY, then G_1 and G_2 form a YES-instance of MUTUAL PLANAR DUALITY. □

3 Decomposition Trees and the SPQR-Tree

A graph is *connected* if there exists a path between any pair of vertices. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1-sets and 2-sets are *cutvertices* and *separation pairs*, respectively. A connected graph is *biconnected* if it does not have a cut vertex and *triconnected* if it does not have a separation pair. The maximal biconnected components of a graph are called *blocks*.

In the following we consider *decomposition trees* of biconnected planar graphs. *SPQR-trees* [5], which can be computed in linear time [7], are a special case of decomposition trees. Let G be a planar biconnected graph and let $\{s, t\}$ be a *split pair*, that is either a separation pair or a pair of adjacent vertices. Let further H_1 and H_2 be two subgraphs of G such that $H_1 \cup H_2 = G$ and $H_1 \cap H_2 = \{s, t\}$. Consider the tree $\mathcal{T}$ consisting of two nodes μ_1 and μ_2 associated with the graphs $H_1 + (s, t)$ and $H_2 + (s, t)$, respectively. For each node μ_i, the graph $H_i + (s, t)$ associated with it is the *skeleton* of μ_i, denoted by $\mathrm{skel}(\mu_i)$, and the special directed edge (s, t) is called *virtual edge*. The edge connecting the nodes μ_1 and μ_2 in $\mathcal{T}$ associates the virtual edge $\varepsilon_1 = (s, t)$ in $\mathrm{skel}(\mu_1)$ with the virtual edge $\varepsilon_2 = (s, t)$ in $\mathrm{skel}(\mu_2)$; we say that ε_1 is the *twin* of ε_2 and vice versa. Moreover, we say that ε_1 in $\mathrm{skel}(\mu_1)$ *corresponds* to the neighbor μ_2 of μ_1. This can be expressed as a bijective map $\mathrm{corr}_\mu \colon E(\mathrm{skel}(\mu)) \to N(\mu)$ for each node μ, where $E(\mathrm{skel}(\mu))$ and $N(\mu)$ denote the set of edges in $\mathrm{skel}(\mu)$ and the set of neighbors of μ in $\mathcal{T}$, respectively. In the example above we have $\mathrm{corr}(\varepsilon_1) = \mu_2$ and $\mathrm{corr}(\varepsilon_2) = \mu_1$ (the subscript of corr is omitted as it is clear from the context).

The above-described procedure is called *decomposition* and can be applied further to the skeletons of the nodes of $\mathcal{T}$, leading to a larger tree with smaller skeletons. The decomposition can be undone by *contracting* an edge in $\mathcal{T}$. Let $\{\mu, \mu'\}$ be an edge in $\mathcal{T}$ and let ε be the virtual edge in $\mathrm{skel}(\mu)$ with $\mathrm{corr}(\varepsilon) = \mu'$ having ε' in $\mathrm{skel}(\mu')$ as twin. The contraction of $\{\mu, \mu'\}$ collapses μ and μ' into a single node with the following skeleton. The skeletons of μ and μ' are *glued* together at the twins ε and ε' according to their orientation, i.e., the sources and targets of ε and ε' are identified with each other, respectively. The resulting virtual edge is removed. Iteratively applying the contraction in $\mathcal{T}$ leads to a tree consisting of a single node μ, whose skeleton is independent from the contraction order. The graph *represented* by $\mathcal{T}$ is $\mathrm{skel}(\mu)$.

A *reversed decomposition tree* is defined as a decomposition tree with the only difference that in the decomposition step one of the two twin edges is reversed and in the contraction step they are glued together oppositely. Note that a reversed decomposition tree can be transformed into an equivalent normal decomposition tree representing the same graph by reversing one virtual edge in each pair of twin edges.

A special decomposition tree is the SPQR-tree. A decomposition tree is an SPQR-tree if each inner node is either an S-, a P-, or an R-node whose skeletons contain only virtual edges forming a cycle, a bunch of at least three parallel edges or a triconnected planar graph, respectively, such that no two S-nodes and no two P-nodes are adjacent. Each leaf is a Q-node whose skeleton consists of two vertices connected by one virtual and one normal edge. The SPQR-tree of a biconnected planar graph is unique up to reversal of twins. We assume that all virtual edges in P-node skeletons are oriented in the same direction and those in S-node skeletons form a directed cycle.

There is a bijection between the embeddings of a biconnected graph G and the set of all combinations of embeddings of the skeletons in its SPQR-tree $\mathcal{T}$. The embedding choices for the skeletons consist of reordering the parallel edges in a P-node and flipping the skeleton of an R-node. Fixing the embeddings of skeletons in an arbitrary decomposition tree $\mathcal{T}$ also determines a planar embedding of the represented graph G. However, there may be planar embeddings that are not represented by $\mathcal{T}$.

We assume the skeletons of the SPQR-tree of a graph to be embedded graphs if and only if the graph itself is embedded.

4 Succinct Representation of all Duals of a Biconnected Graph

Let G be a biconnected graph with SPQR-tree $\mathcal{T}$ and planar embedding $\mathcal{G}$. In the following we study the effects of changing the embedding of G on its dual graph $G^\star$. To this end, we do not consider the graphs themselves but their SPQR-trees. We first show how the SPQR-tree of $G^\star$ can be directly obtained from the SPQR-tree of G.

We first define the *dual decomposition tree* $\mathcal{T}^\star$ of a decomposition tree $\mathcal{T}$ representing G (with respect to a fixed embedding $\mathcal{G}$ of G represented by $\mathcal{T}$). Essentially, $\mathcal{T}^\star$ is obtained from $\mathcal{T}$ by replacing each skeleton with its directed dual and interpreting the resulting tree as a reversed decomposition tree. More precisely, for each node μ in $\mathcal{T}$, the dual decomposition tree $\mathcal{T}^\star$ contains a *dual node* $\mu^\star$ having the dual of $\mathrm{skel}(\mu)$ as skeleton. An edge $\varepsilon^\star$ in $\mathrm{skel}(\mu^\star)$ dual to a virtual edge ε in $\mathrm{skel}(\mu)$ is again virtual and oriented from right to left with respect to the orientation of ε. Two virtual edges in $\mathcal{T}^\star$ are twins if and only if their primal edges are twins. This has the effect that $\mathrm{corr}(\varepsilon)^\star = \mathrm{corr}(\varepsilon^\star)$ holds. In case $\mathcal{T}$ is the SPQR-tree of G, the dual of a triconnected skeleton is triconnected, the dual of a (directed) cycle is a bunch of parallel edges (all directed in the same direction), and the dual of a normal edge with a parallel virtual edge is a normal edge with a parallel virtual edge. Thus, if a node μ in $\mathcal{T}$ is an S-, P-, Q-, or R-node, its dual node $\mu^\star$ in $\mathcal{T}^\star$ is a P-, S-, Q-, or R-node, respectively. Thus, the dual SPQR-tree is again an SPQR-tree and not just an arbitrary decomposition tree.

Lemma 1. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ and embedding $\mathcal{G}$. The dual SPQR-tree $\mathcal{T}^\star$ with respect to $\mathcal{G}$ is the reversed SPQR-tree of the dual $G^\star$.*

Sketch of Proof. We show the claim for general decomposition trees. As illustrated in Figure 2, first contracting an edge $\{\mu, \mu'\}$ in a decomposition tree $\mathcal{T}$ and then taking the dual decomposition tree is equivalent to first taking the dual decomposition tree $\mathcal{T}^\star$ and then contracting $\{\mu^\star, \mu'^\star\}$. Applying this operation iteratively until the trees $\mathcal{T}$ and $\mathcal{T}^\star$ consist of single nodes directly shows that the reversed decomposition tree $\mathcal{T}^\star$ represents the graph $G^\star$ dual to the graph G represented by $\mathcal{T}$. □

In the following we consider how the dual SPQR-tree changes when the embedding of skeletons in the SPQR-tree change. Flipping the skeleton of an R-node and reordering the virtual edges in a P-node give rise to the following two operations: *reversal* of R-nodes and *restacking* of S-nodes. A reversal applied on an R-node μ reverses the direction of all edges in $\mathrm{skel}(\mu)$. Note that this only affects how $\mathrm{skel}(\mu)$ is glued to the skeletons of its adjacent nodes. Let μ be an S-node with virtual edges $\varepsilon_1, \dots, \varepsilon_k$. A restacking of μ picks an arbitrary ordering of $\varepsilon_1, \dots, \varepsilon_k$ and glues their end-points such that they create a directed cycle C in that order. Then, $\mathrm{skel}(\mu)$ is replaced by C.

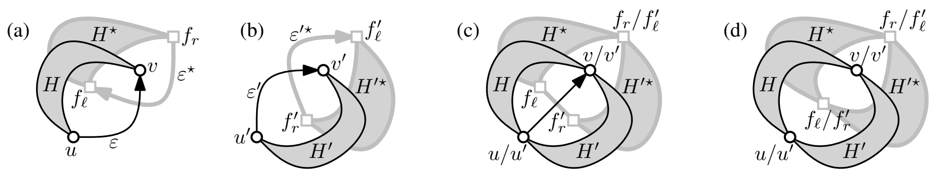


Fig. 2. (a)–(c) Glueing the virtual edge ε and ε'. (d) Removing the resulting edge.

Lemma 2. *Let $\mathcal{T}$ and $\mathcal{T}^\star$ be the SPQR-trees of an embedded biconnected planar graph and of its dual, respectively. Flipping an R-node and reordering a P-node in $\mathcal{T}$ corresponds to reversing its dual R-node and restacking its dual S-node, respectively.*

Proof. Due to Lemma 1 we can work with the dual SPQR-tree instead of the SPQR-tree of the dual. Obviously, flipping an R-node μ in $\mathcal{T}$ exchanges left and right in $\mathrm{skel}(\mu)$ and thus reverses the orientation of each virtual edge in $\mathrm{skel}(\mu^\star)$, where $\mu^\star$ is the node in $\mathcal{T}^\star$ dual to μ. Thus, flipping μ corresponds to a reversal of $\mu^\star$. Similarly, reordering the virtual edges in the skeleton of a P-node μ has the effect that the virtual edges in its dual S-node $\mu^\star$ are restacked, yielding a different cycle. Note that this cycle is again directed since the virtual edges in μ are still all oriented in the same direction. □

This shows that the SPQR-tree of the dual graph with respect to a fixed embedding can be used to represent the dual graphs with respect to all possible planar embeddings by allowing reversal and restacking operations. We say that an SPQR-tree *represents a dual graph* if it can be obtained by applying reversal and restacking operations.

Theorem 3. *The dual SPQR-tree of a biconnected planar graph G represents exactly the dual graphs of G.*

When we are not interested in the embedding of the dual graph but only in its structure, we may also allow the usual SPQR-tree operations, that is flipping R-nodes and reordering the edges in P-nodes. Note that the reversal operation applied to P-nodes only changes the embedding of the graph and not its structure. Moreover, reversing a Q-node does not change anything and the reversal of an S-node can be seen as a special way of restacking it. This observation can be used to show the following lemma.

Lemma 3. *Let G be a biconnected planar graph with embedding $\mathcal{G}$ and let $G^\star$ be its dual graph with SPQR-tree $\mathcal{T}^\star$. Let $\mathcal{T}^\star_\varepsilon$ be the SPQR-tree obtained from $\mathcal{T}^\star$ by reversing the orientation of the virtual edge ε in $\mathcal{T}^\star$ and let $G^\star_\varepsilon$ be the graph it represents. There exists an embedding $\mathcal{G}_\varepsilon$ of G such that $G^\star_\varepsilon$ is the dual graph of G with respect to $\mathcal{G}_\varepsilon$.*

Sketch of Proof. Let μ be the node in $\mathcal{T}^\star$ containing the virtual edge ε and let $\mathrm{corr}(\varepsilon) = \mu'$ be the neighbor of μ corresponding to ε. Removing the edge $\{\mu, \mu'\}$ splits $\mathcal{T}^\star$ into two subtrees $\mathcal{T}^\star_\mu$ and $\mathcal{T}^\star_{\mu'}$. One can show that the reversal of all nodes in one of these subtrees (no matter which one) yields an SPQR-tree $\mathcal{T}^\star_{\mu\mu'}$ representing $G^\star_\varepsilon$. Then it follows by Lemma 2 and the observation above, that $G^\star_\varepsilon$ is a dual graph of G. □
Lemma 2 and Lemma 3 together yield the following theorem.

Theorem 4. *For two SPQR-trees $\mathcal{T}_1$ and $\mathcal{T}_2$, the following three statements are equivalent. 1. $\mathcal{T}_1$ and $\mathcal{T}_2$ represent the same set of dual graphs. 2. $\mathcal{T}_1$ and $\mathcal{T}_2$ can be transformed into each other using reversal and restacking operations. 3. $\mathcal{T}_1$ and $\mathcal{T}_2$ can be transformed one into the other by choosing orientations for the virtual edges and by restacking S-node skeletons.*

5 Equivalence Relation

We define the relation $\sim$ on the set of planar graphs as follows. Two graphs G_1 and G_2 are related, i.e., $G_1 \sim G_2$, if and only if G_1 and G_2 can be embedded such that they have the same dual graph $G_1^\star = G_2^\star$. We call $\sim$ the *common dual relation*.

Theorem 5. *The common dual relation $\sim$ is an equivalence relation on the set of biconnected planar graphs. For a biconnected planar graph G, the set of dual graphs of G is an equivalence class with respect to $\sim$.*

Proof. Clearly, $\sim$ is symmetric and reflexive. For the transitivity let G_1, G_2 and G_3 be three biconnected planar graphs such that $G_1 \sim G_2$ and $G_2 \sim G_3$. Let further $\mathcal{T}_1^\star$, $\mathcal{T}_2^\star$ and $\mathcal{T}_3^\star$ be the dual SPQR-trees representing all duals of G_1, G_2 and G_3, respectively. Due to $G_1 \sim G_2$ there exists a graph G that is represented by $\mathcal{T}_1^\star$ and $\mathcal{T}_2^\star$. Thus, $\mathcal{T}_1^\star$ and $\mathcal{T}_2^\star$ can both be transformed into the SPQR-tree representing G using reversal and restacking operations, which shows that they represent the same set of duals (Theorem 4). The same argument shows that G_2 and G_3 have the same set of dual graphs. Thus, also G_1 and G_3 have exactly the same set of dual graphs, which yields $G_1 \sim G_3$.

For the second statement, let $C^\star$ be the set of dual graphs of G. Clearly, for $G_1^\star, G_2^\star \in C^\star$ the graph G is a common dual, thus $G_1^\star \sim G_2^\star$. On the other hand, let $G_1^\star \in C^\star$ and $G_1^\star \sim G_2^\star$. By the above argument, $G_1^\star$ and $G_2^\star$ have the same set of dual graphs. Thus G is a dual graph of $G_2^\star$ yielding $G_2^\star \in C^\star$. □

Theorem 5 shows that the equivalence class C of a biconnected planar graph G with respect to the common dual relation is exactly the set of dual graphs that is represented by the SPQR-tree $\mathcal{T}$ of G. The dual SPQR-tree $\mathcal{T}^\star$ of G also represents a set of dual graphs forming the equivalence class $C^\star$. We say that $C^\star$ is the *dual equivalence class* of C. Given an arbitrary graph $G \in C$ and an arbitrary graph $G^\star \in C^\star$, graphs G and $G^\star$ can be embedded such that they are dual to each other, since $C^\star$ contains exactly the graphs that are dual to G. The problems MUTUAL PLANAR DUALITY and GRAPH SELF-DUALITY can be reformulated in terms of the equivalence classes of the common dual relation. Two biconnected planar graphs are a YES-instance of MUTUAL PLANAR DUALITY if and only if their equivalence classes are dual to each other. A biconnected planar graph is graph self-dual if and only if its equivalence class is dual to itself. This in particular means that either each or no graph in an equivalence class is graph self-dual.

Although it might seem quite natural that the common dual relation is an equivalence relation, this is not true for general planar graphs, see Figure 3.

Theorem 6. *The common dual relation $\sim$ is not transitive on the set of planar graphs.*

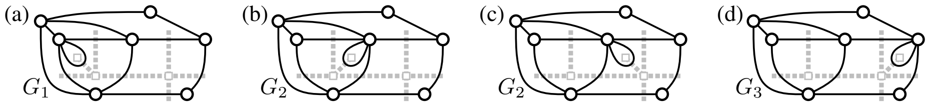


Fig. 3. The graphs G_1 (a) and G_2 (b) have a common dual and the graphs G_2 (c) and G_3 (d) have a common dual. The graphs G_1 and G_3 do not have a common dual.

6 Solving Mutual Planar Duality for Biconnected Graphs

The problem Mutual Planar Duality can be rephrased as follows.

Corollary 1. *Two biconnected planar graphs G_1 and G_2 with SPQR-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ form a* Yes*-instance of* Mutual Planar Duality *if and only if $\mathcal{T}_2$ and the dual SPQR-tree $\mathcal{T}_1^\star$ represent the same dual graphs.*

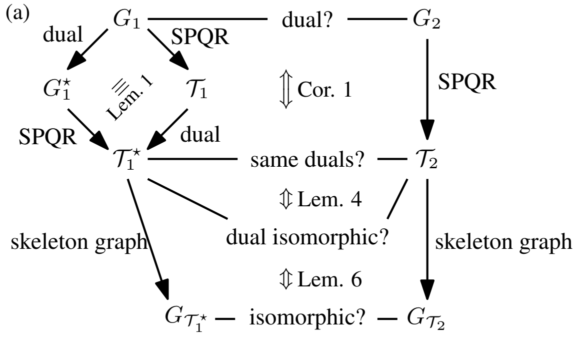
In the following we show that two SPQR-trees represent the same set of dual graphs if and only if they are *dual isomorphic* (we define this in a moment). Then we show that testing the existence of such an isomorphism reduces to testing graph isomorphism for planar graphs. Figure 4(a) sketches this strategy.

For two graphs G and G' with vertices $V(G)$ and $V(G')$ and edges $E(G)$ and $E(G')$, respectively, a map $\varphi\colon V(G) \to V(G')$ is a *graph isomorphism* if it is bijective and $\{u, v\} \in E(G)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(G')$ (for directed graphs, the direction of the edges is disregarded). A graph isomorphism φ induces a bijection between $E(G)$ and $E(G')$ and we use $\varphi(e)$ for $e \in E(G)$ to express this bijection. As we consider undirected edges, fixing $\varphi(\cdot)$ only for the edges is not sufficient. A *dual SPQR-tree isomorphism* between two SPQR-trees $\mathcal{T}$ and $\mathcal{T}'$ consists of several maps. First, a map $\varphi\colon V(\mathcal{T}) \longrightarrow V(\mathcal{T}')$ such that

(I) φ is a graph isomorphism between $\mathcal{T}$ and $\mathcal{T}'$; and
(II) for each node $\mu \in V(\mathcal{T})$, the node $\varphi(\mu) \in V(\mathcal{T}')$ is of the same type.

Second, a map $\varphi_\mu\colon V(\text{skel}(\mu)) \longrightarrow V(\text{skel}(\varphi(\mu)))$ for every R-node μ in $\mathcal{T}$ such that

(III) φ_μ is a graph isomorphism between $\text{skel}(\mu)$ and $\text{skel}(\varphi(\mu))$; and
(IV) $\text{corr}(\varphi_\mu(\varepsilon)) = \varphi(\text{corr}(\varepsilon))$ holds for every virtual edge ε in $\text{skel}(\mu)$.



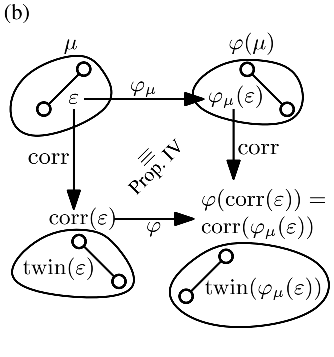

(b)
μ
$\varphi(\mu)$
ε
φ_μ
$\varphi_\mu(\varepsilon)$
corr
Prop. IV
corr
$\varphi(\text{corr}(\varepsilon)) =$
$\text{corr}(\varphi_\mu(\varepsilon))$
$\text{corr}(\varepsilon)$
φ
$\text{twin}(\varepsilon)$
$\text{twin}(\varphi_\mu(\varepsilon))$

Fig. 4. (a) Overview of our strategy. (b) Commutative diagram illustrating Property IV

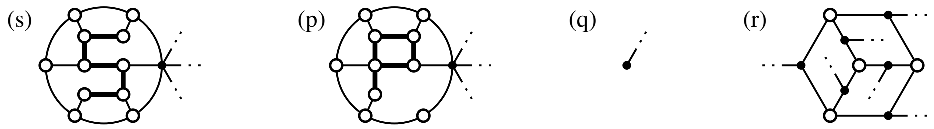


Fig. 5. The subgraphs H_μ of the skeleton graph depending on the type of the node μ. The small black vertices are the attachment vertices.

If there is a dual SPQR-tree isomorphism between $\mathcal{T}$ and $\mathcal{T}'$, then we say that $\mathcal{T}$ and $\mathcal{T}'$ are *dual isomorphic*. Note that Property IV (illustrated in Figure 4(b)) is a natural requirement and one would usually require it also for S-nodes (for P-nodes it does not make sense since every permutation is an isomorphism on its skeleton). However, not requiring it for S-nodes implicitly allows restacking their skeletons. As the graph isomorphisms $\varphi_\mu(\cdot)$ do not care about the orientation of virtual edges, it is also implicitly allowed to reverse them. We get the following lemma showing that this definition of dual SPQR-tree isomorphism is well suited for our purpose.

Lemma 4. *Two SPQR-trees represent the same set of dual graphs if and only if they are dual isomorphic.*

We reduce dual SPQR-tree isomorphism testing to graph isomorphism testing for planar graphs, which can be solved in linear time [8]. We define the *skeleton graph* $G_\mathcal{T}$ of an SPQR-tree $\mathcal{T}$ as follows. For each node μ in $\mathcal{T}$ there is a subgraph H_μ in $G_\mathcal{T}$ and for each edge $\{\mu, \mu'\}$ in $\mathcal{T}$ there is an edge connecting H_μ and $H_{\mu'}$. In the following we describe the subgraphs H_μ for the cases that μ is an S-, P-, Q-, or R-node and define *attachment vertices* that are incident to the edges connecting H_μ to other subgraphs.

If μ is an S- or P-node, the subgraph H_μ contains only one attachment vertex v_μ and all subgraphs representing neighbors of μ are attached to v_μ. To distinguish between S- and P-nodes, small non-isomorphic subgraphs called *tags* are attached to v_μ, see Figure 5(s) and (p). If μ is a Q-node, then H_μ is a single attachment vertex, see Figure 5(q). Note that μ is a leaf in $\mathcal{T}$ and thus H_μ is also a leaf in $G_\mathcal{T}$. If μ is an R-node, H_μ is the skeleton $\mathrm{skel}(\mu)$, where additionally every virtual edge ε is subdivided by an attachment vertex v_ε, see Figure 5(r) for an example. The subgraph $H_{\mathrm{corr}(\varepsilon)}$ stemming from the neighbor $\mathrm{corr}(\varepsilon)$ of μ is attached to H_μ over the attachment vertex v_ε.

Lemma 5. *The skeleton graph is planar and can be computed in linear time.*

Lemma 6. *Two SPQR-trees are dual isomorphic if and only if their skeleton graphs are isomorphic.*

Sketch of Proof. Let φ together with $\varphi_{\mu_1}, \dots, \varphi_{\mu_k}$ be a dual SPQR-tree isomorphism between the SPQR-trees $\mathcal{T}$ and $\mathcal{T}'$. We show how this induces a graph isomorphism φ_G between the skeleton graphs $G_\mathcal{T}$ and $G_{\mathcal{T}'}$. If μ is an S-, P- or Q-node, then its corresponding subgraph in H_μ only contains a single attachment vertex v_μ. Since $\varphi(\mu)$ is of the same type (Property II), the subgraph $H_{\varphi(\mu)}$ also contains a single attachment vertex $v_{\varphi(\mu)}$ and we set $\varphi_G(v_\mu) = v_{\varphi(\mu)}$. For S- and P-nodes we additionally map their tags to each other. If μ is an R-node, the map φ_μ is a graph isomorphism between $\mathrm{skel}(\mu)$ and

$\mathrm{skel}(\varphi(\mu))$ (Property III). Thus, it induces a graph isomorphism between H_μ and $H_{\varphi(\mu)}$ since these subgraphs are obtained from $\mathrm{skel}(\mu)$ and $\mathrm{skel}(\varphi(\mu))$, respectively, by subdividing each virtual edge. Finally, φ_G respects the edges between attachment vertices of different subgraphs, since φ maps adjacent nodes to each other (Property I) and since these edges connect the correct attachment vertices of the subgraphs (Property IV).

We only sketch the opposite direction. Assume φ_G is a graph isomorphism between $G_\mathcal{T}$ and $G_{\mathcal{T}'}$. As bridges are mapped to bridges, we directly get an isomorphism φ between the trees $\mathcal{T}$ and $\mathcal{T}'$. As leaves have to be mapped to leaves, Q-nodes are mapped to Q-nodes. Moreover, the tags ensure that other nodes are mapped to nodes of the same type. Thus, Properties I and II are satisfied. Moreover, for every R-node μ in $\mathcal{T}$, φ_G induces an isomorphism φ_μ between μ and $\varphi(\mu)$ satisfying Properties III and IV. $\square$

Following the outline given in Fig. 4(a) thus reduces MUTUAL PLANAR DUALITY for biconnected graphs to planarity testing for planar graph, which is linear-time solvable [8].

Theorem 7. MUTUAL PLANAR DUALITY *is linear-time solvable for biconnected graphs.*

Corollary 2. GRAPH SELF-DUALITY *is linear-time solvable for biconnected graphs.*

References

1. Angelini, P., Di Battista, G., Patrignani, M.: Finding a minimum-depth embedding of a planar graph in $O(n^4)$ time. Algorithmica 60, 890–937 (2011)
2. Angelini, P., Bläsius, T., Rutter, I.: Testing mutual duality of planar graphs. CoRR abs/1303.1640 (2013)
3. Archdeacon, D., Richter, R.B.: The construction and classification of self-dual spherical polyhedra. Journal of Combinatorial Theory, Series B 54(1), 37–63 (1992)
4. Bienstock, D., Monma, C.: On the complexity of embedding planar graphs to minimize certain distance measures. Algorithmica 5, 93–109 (1990)
5. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica 15(4), 302–318 (1996)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
7. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
8. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC 1974), pp. 172–184. ACM (1974)
9. Raghavendra Rao, B., Jayalal Sarma, M.: On the complexity of matroid isomorphism problem. Theory of Computing Systems 49(2), 246–272 (2011)
10. Servatius, B., Christopher, P.R.: Construction of self-dual graphs. Am. Math. Monthly 99(2), 153–158 (1992)
11. Servatius, B., Servatius, H.: Self-dual graphs. Discrete Math. 149(1-3), 223–232 (1996)
12. Tutte, W.T.: Connectivity in matroids. Canad. J. Math. 18, 1301–1324 (1966)
13. Whitney, H.: Congruent graphs and the connectivity of graphs. Amer. J. Math. 54(1), 150–168 (1932)
14. Whitney, H.: 2-isomorphic graphs. Amer. J. Math. 55, 245–254 (1933)

Effective and Efficient Data Reduction for the Subset Interconnection Design Problem*

Jiehua Chen[1], Christian Komusiewicz[1], Rolf Niedermeier[1], Manuel Sorge[1], Ondřej Suchý[2], and Mathias Weller[3]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin
{jiehua.chen,christian.komusiewicz, rolf.niedermeier,manuel.sorge}@tu-berlin.de
[2] Department of Theoretical Computer Science, Czech Technical University in Prague
ondrej.suchy@fit.cvut.cz
[3] Département Informatique, LIRMM
mathias.weller@lirmm.fr

Abstract. The NP-hard Subset Interconnection Design problem is motivated by applications in designing vacuum systems and scalable overlay networks. It has as input a set V and a collection of subsets $V_1, V_2, \ldots, V_m$, and asks for a minimum-cardinality edge set E such that for the graph $G = (V, E)$ all induced subgraphs $G[V_1], G[V_2], \ldots, G[V_m]$ are connected. It has also been studied under the name Minimum Topic-Connected Overlay. We study Subset Interconnection Design in the context of polynomial-time data reduction rules that preserve optimality. Our contribution is threefold: First, we point out flaws in earlier polynomial-time data reduction rules. Second, we provide a fixed-parameter tractability result for small subset sizes and tree-like output graphs. Third, we show linear-time solvability in case of a constant number m of subsets, implying fixed-parameter tractability for the parameter m. To achieve our results, we elaborate on polynomial-time data reduction rules (partly "repairing" previous flawed ones) which also may be of practical use in solving Subset Interconnection Design.

1 Introduction

We study relevant tractable cases of the following NP-complete decision problem:

Subset Interconnection Design (SID)
Input: A hypergraph $H = (V, \mathcal{F})$, $k \in \mathbb{N}$.
Question: Is there a graph $G = (V, E)$ such that $|E| \leq k$ and for each $F \in \mathcal{F}$ the induced subgraph $G[F]$ is connected?

Throughout this work, we refer to graphs G in which $G[F]$ is connected for each $F \in \mathcal{F}$ as *solutions*. Solutions with a minimum number of edges are called

* JC was supported by Studienstiftung des Deutschen Volkes, MS and MW were supported by Deutsche Forschungsgemeinschaft (projects NI 369/12 and NI 369/9), and part of the work of OS and MW was done while they were affiliated with TU Berlin.

optimal. Although we present our results for the decision version of the problem, our positive algorithmic results can be easily adapted to its optimization version.

SID has applications in the design of vacuum systems [5, 6], in the design of scalable overlay networks [2, 11, 14], in the design of reconfigurable interconnection networks [7, 8], and in inferring a most likely social network [1]. Indeed, the respective research communities seemed largely unaware of each other's work, for instance leading to multiple NP-hardness proofs. Du [4] seemed to be the first to have formally defined the problem and claimed NP-hardness; to the best of our knowledge, the first published NP-hardness proof is due to Du and Miller [6]. SID has been independently studied under the name Minimum Topic-Connected Overlay by the "scalable overlay networks community" [2, 11, 14] and under the name Interconnection Graph Problem by the "reconfigurable interconnection systems community" [7, 8]. Moreover, the "social network inference community" [1], who additionally imposes edge costs, refers to this more general problem as Network Inference. The term "topic-connected" in Minimum Topic-Connected Overlay refers to the desired property of overlay networks that agents interested in some particular topic should be able to inform each other about updates concerning this topic without involving other agents [14].

Our main focus is on the problem-specific parameters "size $d := \max_{F \in \mathcal{F}} |F|$ of the largest hyperedge" and "number m of hyperedges" in the given hypergraph H. We perform a parameterized complexity analysis with respect to these parameters. Notably, we always have $d \leq k + 1$, where k is the number of edges of the constructed solution. In particular, our core working machinery is the development of numerous polynomial-time data reduction rules, thereby extending and improving some previous work. We use n to denote the number $|V|$ of vertices in the input hypergraph and $|H|$ to denote $\sum_{F \in \mathcal{F}} |F|$.

Previous Results. As mentioned before, SID has been independently studied in different communities. Several NP-hardness proofs have appeared [2, 6, 7].[1] NP-hardness even holds for hypergraphs with $d = 3$ [8, 11], while $d \leq 2$ allows for polynomial-time solvability [11]. There also has been intense study of the polynomial-time approximability, providing various logarithmic-factor approximation algorithms [1, 2, 11] and inapproximability results (implying that logarithmic-factor approximation algorithms are optimal) [1, 11]. The currently best exact algorithm for SID has a running time of $O(n^{2k}/4^k + n^2)$ [11]. In addition, in a series of papers it has been shown that SID can be solved in polynomial time if $2 \leq m \leq 4$ [4, 15, 16]. A variant of SID where the edges incur costs and where the solution is restricted to be a tree has been studied in the context of communication network design; three variations of this tree-construction problem have been shown polynomial-time solvable [12]. Finally, we mention in passing that in the context of overlay networks it is of specific

[1] The reduction in [2] actually only shows NP-hardness of the problem aiming to minimize the maximum degree of a solution.

interest to search for solutions with small maximum and small average vertex degree [2, 14]; the latter is achieved by SID.

Our Contributions. We start by revealing a serious bug in "plausible" data reduction rules (two very similar rules) used in previous work [8, 11], constructing a counterexample showing their incorrectness. Based on this, we provide refined and completely new data reduction rules, assuring their correctness and effectiveness. Almost all of our data reduction rules work in a parameter-independent fashion. Making decisive use of the developed data reduction rules, we show that SID can be solved in $d^{O(df)} \cdot \text{poly}(|H|)$ time, where f denotes the size of a minimum feedback edge set of an optimal solution G, that is, the minimum number of edges whose removal makes G acyclic. Our result shows that SID becomes tractable if the solution is required to be almost a tree (compare this with the tree requirement in related work [12]). Furthermore, a simple calculation shows that whenever $f \leq (n-1)/9d$ the exponential term in our algorithm is smaller than the one in the $O(n^{2k}/4^k + n^2)$-time algorithm given by Hosoda et al. [11]. In case that $d \leq 4$ we further show that SID can be reduced in polynomial time to an equivalent instance of $O(f)$ vertices, known as "polynomial-size problem kernel" in parameterized algorithmics. Finally, improving and generalizing previous work [4, 15, 16], we show that SID can be solved in linear time if the input hypergraph contains only a constant number of hyperedges. This implies that SID is fixed-parameter tractable with respect to the parameter m. Due to lack of space, most proofs are deferred to a full version of the paper.

2 Preliminaries

The concept of parameterized complexity was pioneered by Downey and Fellows [3] (see also [9, 13]). A parameterized problem is a language $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is an alphabet. The second component is called the *parameter* of the problem. Typically, the parameter or the "combined" ones are non-negative integers. A parameterized problem L is *fixed-parameter tractable* (fpt) if there is an algorithm that decides whether $(x, k) \in L$ in $g(k) \cdot |x|^{O(1)}$ time, where g is an arbitrary computable function depending only on k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction* [10]. Here, the goal is to transform a given problem instance (x, k) in polynomial time into an equivalent instance (x', k') with parameter $k' \leq k$ such that the size of (x', k') is upper-bounded by some function g only depending on k. If this is the case, we call the instance (x', k') a (problem) *kernel* of size $g(k)$.

The data reduction is usually presented as a series of *reduction rules*, that is, polynomial-time algorithms that take as input an instance of some decision problem and also produces one as output. A reduction rule is *correct* if for each input instance I, the corresponding output instance of the rule is a yes-instance if and only if I is a yes-instance. Search tree algorithms can be described by *branching rules* that reduce one instance of a problem to several instances of the same problem; a branching rule is *correct* if the original instance is a yes-instance if and only if at least one of the constructed instances is a yes-instance.

Let V be a set and $\mathcal{F}$ be a family of subsets of V. We call $H = (V, \mathcal{F})$ a *hypergraph* with *vertex set* V and *hyperedge set* $\mathcal{F}$. Unless stated otherwise, we assume all hypergraphs to not contain singleton hyperedges, empty hyperedges or multiple copies of the same hyperedge since they are not meaningful for SID, and searching for and removing them can be done without increasing our running times. We call $v \in V$ and $F \in \mathcal{F}$ *incident* if $v \in F$. We denote by $\mathcal{F}(v)$ the set of all hyperedges that are incident with v. If $u, v \in V$ and $\mathcal{F}(v) \subseteq \mathcal{F}(u)$ then we say that u *covers* v. Vertices that cover each other are called *twins*; a maximal set of twins is called *twin class*. The *subhypergraph induced* by V' is the hypergraph $H[V'] := (V', \mathcal{F}')$ where $\mathcal{F}' = \{F \in \mathcal{F} \mid F \subseteq V'\}$. By *removing* a vertex v from H, we mean taking the hypergraph $H' = (V \setminus \{v\}, \{F \setminus \{v\} \mid F \in \mathcal{F}\})$. A *hyperwalk* is an alternating sequence of vertices and hyperedges starting and ending with a vertex and such that succeeding elements are incident with each other. A hypergraph is *connected* if there is a hyperwalk between every pair of vertices.

For graphs $G = (V, E)$ with vertex set V and edge set E, we use $E(G)$ to denote the edge set E of graph G. We denote by $G[V']$ the *subgraph of* G *induced* by V'. We also use $G - V'$ as a shorthand for $G[V \setminus V']$. The *feedback edge set* of a graph G is a minimum-size set of edges whose removal makes G a forest. If G is connected, then the size of a feedback edge set is $|E| - |V| + 1$.

3 Fundamental Observations

In this section, we show that a previously proposed data reduction rule for SID is incorrect. We also show some properties of SID and some data reduction rules that are used in our algorithms.

A very natural approach to identify edges of optimal solutions is to look for vertices u and v such that u covers v, that is, $\mathcal{F}(v) \subseteq \mathcal{F}(u)$. The following shows that degree-one vertices of the solution are adjacent to vertices that cover them.

Observation 1. If the hypergraph $H = (V, \mathcal{F})$ has a solution G such that some $u \in V$ has only one neighbor v in G, then v covers u.

It is thus tempting to devise a reduction rule that adds an edge between such vertices: creating a degree-one vertex should be optimal since every vertex needs at least one incident edge. Indeed, such a reduction rule was proposed for vertex pairs u, v that are twins, that is, they are in the same hyperedges [8], or where one covers the other [11]. The variant of these reduction rules that is applicable less often reads as follows.

Rule 1. If vertices u and v are twins, that is $\mathcal{F}(u) = \mathcal{F}(v)$, then remove u from H and decrease k by one.

Unfortunately, this rule is not correct, as a counterexample shows.

Lemma 1. *There is a yes-instance* $(H = (V, \mathcal{F}), k)$ *containing twins* u *and* v *such that Rule 1 applied to* u *and* v *yields a no-instance.*

Proof. Let $f \geq 3$ be an arbitrary integer. Consider the hypergraph $H = (V, \mathcal{F})$, with vertex set $V = \{u, v, a_1, \dots, a_f, b_1, \dots, b_f\}$ and hyperedge set $\mathcal{F}$ which is the union of the following sets of hyperedges:

$$\begin{aligned}
&\mathcal{F}_1 = \{\{a_i, b_i\} \mid i \in \{1, \dots, f\}\},\ \mathcal{F}_2 = \{\{u, v, a_i, b_i\} \mid i \in \{1, \dots, f\}\},\\
&\mathcal{F}_3 = \{\{u, v, a_i, b_i, a_j\} \mid i, j \in \{1, \dots, f\}, i \neq j\}, \text{ and}\\
&\mathcal{F}_4 = \{\{u, v, a_i, b_i, b_j\} \mid i, j \in \{1, \dots, f\}, i \neq j\}.
\end{aligned}$$

Note that the graph $G = (V, E)$ with $E := \mathcal{F}_1 \cup \{\{a_i, u\}, \{b_i, v\} \mid i \in \{1, \dots, f\}\}$ is a solution for H containing $3f$ edges. Hence, $(H, 3f)$ is a yes-instance.

Now, let $(H' = (V', \mathcal{F}'), 3f-1)$ be an instance that results from $(H, 3f)$ by applying Rule 1 to u and v, that is, removing u from H and decreasing the solution size by one. Then, $V' = V \setminus \{u\}$ and $\mathcal{F}'$ consists of the following hyperedges:

$$\begin{aligned}
&\mathcal{F}_1 = \{\{a_i, b_i\} \mid i \in \{1, \dots, f\}\},\ \mathcal{F}_2' = \{\{v, a_i, b_i\} \mid i \in \{1, \dots, f\}\},\\
&\mathcal{F}_3' = \{\{v, a_i, b_i, a_j\} \mid i, j \in \{1, \dots, f\}, i \neq j\}, \text{ and}\\
&\mathcal{F}_4' = \{\{v, a_i, b_i, b_j\} \mid i, j \in \{1, \dots, f\}, i \neq j\}.
\end{aligned}$$

We show that every solution for H' has at least $3f$ edges and, thus, $(H', 3f-1)$ is a no-instance. First, every solution for H' contains the f edges corresponding to the size-two hyperedges of $\mathcal{F}_1$. Furthermore, due to the hyperedges in $\mathcal{F}_2'$, for each $i \in \{1, \dots, f\}$, either $\{v, a_i\}$ or $\{v, b_i\}$ is in any solution. By the symmetry between a_i and b_i in the created hypergraph, assume without loss of generality that an optimal solution contains the edge $\{v, b_i\}$ for all $i \in \{1, \dots, f\}$. Now, let $G' = (V', E')$ be such a solution for H' and let $A_1 = \{a_i \mid \{v, a_i\} \notin E'\}$ be the set of a_is that are *not* adjacent to v in G' and let A_2 denote the remaining a_is. Now if $A_1 = \emptyset$, then G' contains at least $3f$ edges. We show that in case $A_1 \neq \emptyset$ the graph G' also has at least $3f$ edges. Assume that G' is optimal and that every optimal solution has at least $g > 0$ vertices in A_1. For every hyperedge $F = \{v, a_i, b_i, a_j\}$ with $a_j \in A_1$ and $i \in \{1, \dots, f\} \setminus \{j\}$, G' has an edge between a_j and $\{v, a_i, b_i\}$ since $G'[F]$ is connected. Note that if G' contains the edge $\{b_i, a_j\}$, then we can replace this edge by $\{v, a_j\}$: The hyperedge F is the only hyperedge that contains $\{b_i, a_j\}$ and does not already induce a connected subgraph. Clearly, $G'[F]$ can also be made connected by adding $\{v, a_j\}$ instead. This implies an optimal solution with $g - 1$ vertices in A_1, contradicting our choice of g. Hence, G' contains no edges $\{b_i, a_j\}$ with $i \neq j$. Consequently, in order to make each $\{v, a_i, b_i, a_j\} \in \mathcal{F}_4'$ with $a_j \in A_1$ connected, there is an edge between a_i and a_j.

Hence, G' has $g \cdot (f-g)$ edges between A_1 and A_2, $\binom{g}{2}$ edges between vertices in A_1 and another $f - g$ edges between v and A_2. Altogether the total number of edges in G' is thus at least $2f + g \cdot (f - g) + \binom{g}{2} + f - g \geq 3f$. This implies that $(H', 3f - 1)$ is a no-instance. □

With some additional conditions, rules similar to Rule 1 are correct (Rules 2 to 4, 6, and 8 below). First, if a vertex u is adjacent to some vertex v covering u in an optimal solution, then there is an optimal solution that shifts some or all other edges incident with u to v.

Lemma 2. *Let u, v be two vertices in a hypergraph H with v covering u. If H has an optimal solution G containing the edge $\{u, v\}$, then H also has an optimal solution with u being adjacent only to v.*

The above lemma immediately implies the following reduction rule.

Rule 2. If hypergraph H contains vertices u, v such that v covers u and there is an optimal solution G containing the edge $\{u, v\}$, then remove u from H and decrease k by one.

Note that the correctness of Rule 2 together with Lemma 1 implies that there are instances in which twins or vertices that cover each other are *not* adjacent in any optimal solution.

In the counterexample to Rule 1, there are only two twins and they are contained in hyperedges of size five, that is, the size-five hyperedges containing these two vertices have three other "unrelated" vertices. In the following, we show that this is tight, that is, if in each hyperedge that contains some u, all except two unrelated vertices cover u, then the reduction rule is correct.

Rule 3. If there are vertices u and $v_1, \ldots, v_q$ such that $\mathcal{F}(u) \subseteq \mathcal{F}(v_i)$ for every $i \in \{1, \ldots, q\}$ and for each hyperedge $F \in \mathcal{F}(u)$ we have $|F| \leq q + 3$, then remove u from H and decrease k by one.

Lemma 3. *Rule 3 is correct and can be applied exhaustively in $O(n \cdot |H|)$ time.*

Proof (Sketch). Let $Q = \{v_1, \ldots, v_q\}$ and N the set of neighbors of u in an optimal solution G. If $N \cap Q \neq \emptyset$ then the correctness follows from Rule 2. Otherwise, if N contains a neighbor w of some $v \in Q$ in G, then removing $\{u, w\}$ and adding $\{u, v\}$ yields another optimal solution and we can apply Rule 2. If N contains no neighbor of any $v \in Q$ we obtain $|F \cap N| \leq 1$ for all $F \in \mathcal{F}(u)$ because of the size bound on F. Hence, removing all edges incident with u and adding to u a single edge to a vertex in Q does not disconnect any $F \in \mathcal{F}(u)$.

The running time proof is deferred to a full version of the paper. □

As a corollary of Lemma 3, we also obtain correctness of the following rule since it is a special case of Rule 3. This rule will be useful in the next section.

Rule 4. If there are two vertices u and v such that $\mathcal{F}(u) \subseteq \mathcal{F}(v)$ and $|F| \leq 4$ for each hyperedge $F \in \mathcal{F}(u)$, then remove u from H and decrease k by one.

Note that the condition $|F| \leq 4$ in Rule 4 is also tight in the sense that if u is incident with hyperedges of size at least five, this rule is not correct (Lemma 1).

4 Data Reduction Rules for Sparse Solutions

In this section, we present a set of reduction rules whose aim is to remove parts of the instance where optimal solutions can be identified in polynomial time. In particular, we aim at finding structures that either produce tree-like parts or long degree-two paths in the solution. We stress that our data reduction rules are applicable regardless of the structure of an optimal solution. We merely use the size of its feedback edge set to provide formal performance guarantees.

4.1 Problem Kernel for f and $d \leq 4$

We now describe how we can remove all but $O(f)$ vertices from a SID instance with $d \leq 4$ in $O(n \cdot m^3)$ time by using Rule 4 and an additional reduction rule. Basically, the parameter f upper-bounds the number of vertices that are in cycles and have degree at least three, while Rule 4 ensures that there are no degree-one vertices in solutions. To get an upper bound on the number of vertices, we also have to deal with long paths. This is the purpose of Rule 5, which is also needed in Section 4.2 to deal with larger hyperedges. Hence, this rule is more general than needed for $d \leq 4$.

Rule 5. Let $(H = (V, \mathcal{F}), k)$ be an instance of SID. If H contains a vertex set $P := \{p_0, \ldots, p_{2d}\}$ with incident hyperedge set $\mathcal{F}' := \bigcup_{p \in P} \mathcal{F}(p)$ such that

1. no $p_i \in P$ covers any $p_j \in P$ with $j \neq i$,
2. for each $F \in \mathcal{F}'$ we have $F \cap P = \{p_i, \ldots, p_j\}$ for some $0 \leq i \leq j \leq 2d$,
3. for each $F \in \mathcal{F}'$ with $F \cap \{p_0, p_{2d}\} = \emptyset$, and for every vertex $v \in F \setminus P$, there is a vertex $p \in P$ that covers v, and
4. there is no hyperedge $F \in \mathcal{F}$ such that $F \cap P = \{p_i\}$ for any $0 < i < 2d$,

then for every $F \in \mathcal{F}'$ with $F \cap \{p_0, p_{2d}\} = \emptyset$, remove all vertices in $F \setminus P$ from H and decrease k by their number. Furthermore, remove the vertices $p_2, \ldots, p_{2d-2}$ from H and decrease k by $2d - 2$.

Intuitively, Conditions 1 and 2 indicate that a solution for such a hypergraph contains a long path and Condition 3 ensures that all vertices not in the path can be attached to it in a simple way.

Next, we give two observations that we need in the correctness proof and in the analysis of the running time of Rule 5. The first observation is about the structure of the hyperedges along the presumed path containing P.

Observation 2. Let H be a hypergraph and $P \subseteq V$ as in Rule 5. For every $0 < i < 2d$ there is a hyperedge F_i^+ such that $p_{i-1} \notin F_i^+$ and $\{p_i, p_{i+1}\} \subseteq F_i^+$ and also a hyperedge F_i^- such that $\{p_{i-1}, p_i\} \subseteq F_i^-$ and $p_{i+1} \notin F_i^-$. Moreover, there is a hyperedge F_0^- such that $F_0^- \cap P = \{p_0\}$ and a hyperedge F_{2d}^+ such that $F_{2d}^+ \cap P = \{p_{2d}\}$.

The second observation provides a lower bound for the number of edges in solutions for connected subhypergraphs.

Observation 3. Let $H = (V, \mathcal{F})$ be a hypergraph and let G be a solution for H. If the subhypergraph $H[V']$ induced by a vertex subset $V' \subseteq V$ is connected, then $|E(G[V'])| \geq |V'| - 1$.

Using these observations we can prove the correctness of Rule 5.

Lemma 4. *Rule 5 is correct and it is possible to find an application of Rule 5 or to decide that it does not apply to the hypergraph in $O(m^3 d^3)$ time.*

We now derive an upper bound on the number of vertices in reduced instances. As mentioned before, we use Rule 5 in Section 4.2, where we also need a similar upper bound on the number of vertices. However, the preconditions of Rule 5 will be satisfied here by Rule 4 and later by a different rule. Hence, we introduce a "cleared"-notion for hypergraphs that will be ensured by these rules. To state our results conveniently, we first introduce another definition.

Definition 1. *The* 2-core *of a graph G is the uniquely defined induced subgraph of G with maximum number of vertices and minimum vertex-degree two.*

Definition 2. *We say that a hypergraph $H = (V, \mathcal{F})$ is* cleared *if there is an optimal solution G for H such that each vertex of degree at least two is in the 2-core of G and, furthermore, for each $P := \{p_0, \ldots, p_{2d}\}$ with $P \subseteq V$ and $\mathcal{F}' := \bigcup_{p \in P} \mathcal{F}(p)$ that satisfy Conditions 1, 2, and 3 of Rule 5, it holds that H and P also satisfy Condition 4.*

It turns out that Rule 4 "clears hypergraphs":

Lemma 5. *Let $H = (V, \mathcal{F})$ be a hypergraph with $d \leq 4$ that is reduced with respect to Rule 4. Then, H is cleared.*

We now bound the size of reduced instances. We also use this bound in Section 4.2 and, hence, prove it in a slightly more general form than needed for $d \leq 4$.

Lemma 6. *Let (H, k) be a yes-instance of* SID *such that H is connected, cleared, and reduced with respect to Rule 5. Then, there is a solution $G = (V, E)$ for (H, k) such that the 2-core of G has at most $(9d - 1)(f - 1)$ vertices and, hence, at most $9d \cdot f$ edges.*

Using Lemmas 5 and 6, and combining them with the observation that hypergraphs that are reduced with Rule 4 have solutions without degree-one vertices, we now obtain that exhaustively applying Rules 4 and 5 yields a polynomial-size problem kernel for SID parameterized by the parameter f, when $d \leq 4$.

Theorem 1. *An instance of* SID *with $d \leq 4$ can be reduced to an equivalent one with at most $35(f - 1)$ vertices in $O(n \cdot m^3)$ time.*

4.2 A Fixed-Parameter Algorithm for f and d

Our polynomial-time data reduction in the last section does not generalize easily to arbitrary d, but, using an additional reduction rule, we can obtain the same vertex-bound of the 2-core of a solution. However, many degree-one vertices may still remain and it seems unclear how to remove them for $d \geq 5$.

Nevertheless, using the bounded 2-core in solutions, we obtain a branching algorithm with running time $O(d^{O(d \cdot f)} \cdot m^2 + n \cdot m^3 \cdot d^3)$. The algorithm first applies Rule 5 and Rule 6 (below) to simplify the structure of the solution that we are looking for. Then, we apply a branching rule that branches into $O(d^2)$ cases and finds at least one of the edges in the 2-core of a solution. If the branching rule does not apply, then an optimal solution can be found in polynomial time.

First, to obtain the bound on the 2-core, we replace Rule 4 with Rule 6 to clear the input hypergraph and to make Lemma 6 applicable.

Rule 6. Let $H = (V, \mathcal{F})$ be a hypergraph and $\{u, u_1, \ldots, u_\ell\} \in \mathcal{F}$ such that u covers each u_i. Then, remove the vertices $u_1, \ldots, u_\ell$ from H and decrease k by ℓ.

We use Rule 6 to replace Rule 4 in clearing hypergraphs.

Lemma 7. *Let $H = (V, \mathcal{F})$ be a hypergraph that is reduced with respect to Rule 6. Then, H is cleared.*

Now, Lemma 6 is applicable to hypergraphs that are reduced with respect to Rule 6 giving us that there is a solution with at most $9d \cdot f$ edges in the 2-core. Based on this lemma, we devise a branching algorithm for the parameter (d, f). This algorithm creates a search tree where at each node of the search tree the current instance consists of a hypergraph H, a partial solution G, and an integer k'. The task is to find a solution G' such that G' is a supergraph of G, all edges of G are within the 2-core of G', and the 2-core of G' has at most k' edges more than G. In order to obtain a search tree whose size depends only on d and f, we ensure that the search tree has depth at most $9d \cdot f$ and that the algorithm branches into at most $\binom{d}{2}$ cases in each step.

In the following, we assume that G and H are reduced with respect to Rule 6.

Branching Rule 1. Let F be a hyperedge of H such that $G[F]$ is disconnected and let $F_0 \subseteq F$ denote the vertices in F that have degree zero in G. Furthermore, $G[F]$ cannot be made connected by adding for each $u \in F_0$ an edge between u and some vertex $v \in F \setminus F_0$ that covers u. Then, branch into all possibilities to add an edge to $G[F]$, decreasing k by one.

Next, we show that, if Branching Rule 1 does not apply to any vertex, then we can solve the instance by greedily assigning the remaining vertices.

Lemma 8. *Let H be a hypergraph and let G be a graph such that there is a solution for H that is a supergraph of G and Branching Rule 1 does not apply to H and G. Then, an optimal solution for H can be computed in $O(n \cdot m)$ time.*

Combining all of the above, we arrive at the main result of this section.

Theorem 2. SID *can be solved in $O(d^{O(d \cdot f)} \cdot m^2 + n \cdot m^3 \cdot d^3)$ time.*

5 Data Reduction for Instances with Few Hyperedges

In this section, we show that SID is fixed-parameter tractable with respect to the number m of hyperedges. A previous fixed-parameter tractability result for this parameter relied on Rule 1 [11, Theorem 8] and is therefore incorrect. In order to restore this result, we need a slightly more involved rule whose correctness proof makes use of the following upper bound on the number of edges needed in the solution.

Lemma 9. *Every instance with $k \geq \binom{2^m}{2} + n$ is a yes-instance.*

The upper bound provided by Lemma 9 grows exponentially in the number of hyperedges. For many purposes, it would be practical to replace this exponential dependence by a polynomial function. However, we note that there are instances that require a solution with at least $n + 2^{\Omega(m)}$ edges (proof deferred).

Lemma 9 directly yields the following reduction rule.

Rule 7. If $k \geq \binom{2^m}{2} + n$, then answer "yes".

The following rule removes vertices from large twin classes.

Rule 8. Let H be an instance that is reduced with respect to Rule 7. If there is a twin class T in H with $|T| > 4^m + 7 \cdot 2^m + 1$, then remove an arbitrary vertex $v \in T$ from H and decrease k by one.

To prove the correctness of Rule 8, we need to show that there is a solution G that has the following property concerning its low-degree vertices.

Lemma 10. *Let $H = (V, \mathcal{F})$ be a hypergraph. There is a solution $G = (V, E)$ such that for each twin class T of H the graph G has*

1. *at most one vertex $t \in T$ that has degree-one neighbors, and*
2. *at most one degree-two vertex $t' \in T$.*

Now, the main idea for the proof of correctness of Rule 8 is to show that a solution G with $k < \binom{2^m}{2} + n$ edges for a hypergraph H cannot contain too many vertices of degree at least three. As a consequence, at least one vertex of T has degree one in G and can be removed safely.

Exhaustive application of Rule 7 and Rule 8 yields a problem kernel for SID parameterized by the number m of hyperedges. Moreover, this kernel can be computed in linear time.

Theorem 3. *An instance of* SID *can be reduced to an equivalent one of size at most $O(8^m \cdot m)$ in $O(|H|)$ time.*

6 Conclusion

Our work leads to a number of interesting tasks for future research: We left open the existence of a polynomial-size problem kernel for Subset Interconnection Design parameterized by the number m of hyperedges; we conjecture that there is none, however. Further, we did not resolve whether SID is fixed-parameter tractable with respect to the feedback edge set size of the solution *alone*. It would also be interesting to significantly improve on the straightforward exponential upper bound $2^{O(n^2)}$ when solving Subset Interconnection Design parameterized by the number n of vertices. It seems also promising to consider data reduction for the variant of Subset Interconnection Design that asks to minimize the maximum degree instead of the average degree (see Onus and Richa [14]). It is furthermore of practical interest to deal with edge weights for the constructed network [12]; our methods only cover the unweighted case. Given the numerous applications, an in-depth investigation of all relevant parameters motivated by real-world instances, that is, performing a parameter analysis for real-world instances, is promising from a practical and from a theoretical side.

Acknowledgment. We thank Peter Damaschke for stimulating discussions and for pointing us to the SUBSET INTERCONNECTION DESIGNS problem.

References

[1] Angluin, D., Aspnes, J., Reyzin, L.: Inferring social networks from outbreaks. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS (LNAI), vol. 6331, pp. 104–118. Springer, Heidelberg (2010)
[2] Chockler, G., Melamed, R., Tock, Y., Vitenberg, R.: Constructing scalable overlays for pub-sub with many topics. In: Proc. 26th PODC, pp. 109–118. ACM (2007)
[3] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
[4] Du, D.-Z.: An optimization problem on graphs. Discrete Appl. Math. 14(1), 101–104 (1986)
[5] Du, D.-Z., Kelley, D.F.: On complexity of subset interconnection designs. J. Global Optim. 6(2), 193–205 (1995)
[6] Du, D.-Z., Miller, Z.: Matroids and subset interconnection design. SIAM J. Discrete Math. 1(4), 416–424 (1988)
[7] Fan, H., Wu, Y.-L.: Interconnection graph problem. In: Proc. FCS 2008, pp. 51–55. CSREA Press (2008)
[8] Fan, H., Hundt, C., Wu, Y.-L., Ernst, J.: Algorithms and implementation for interconnection graph problem. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 201–210. Springer, Heidelberg (2008)
[9] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
[10] Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
[11] Hosoda, J., Hromkovič, J., Izumi, T., Ono, H., Steinová, M., Wada, K.: On the approximability and hardness of minimum topic connected overlay and its special instances. Theor. Comput. Sci. 429, 144–154 (2012)
[12] Korach, E., Stern, M.: The clustering matroid and the optimal clustering tree. Math. Program. 98(1-3), 385–414 (2003)
[13] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
[14] Onus, M., Richa, A.W.: Minimum maximum-degree publish-subscribe overlay network design. IEEE/ACM Trans. Netw. 19(5), 1331–1343 (2011)
[15] Tang, T.-Z.: An optimality condition for minimum feasible graphs. Applied Mathematics - A Journal of Chinese Universities, 24–21 (1989) (in Chinese)
[16] Xu, Y., Fu, X.: On the minimum feasible graph for four sets. Applied Mathematics - A Journal of Chinese Universities 10, 457–462 (1995)

Myhill-Nerode Methods for Hypergraphs

René van Bevern[1], Michael R. Fellows[2],
Serge Gaspers[3], and Frances A. Rosamond[2]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
rene.vanbevern@tu-berlin.de

[2] School of Engineering and IT, Charles Darwin University, Darwin, Australia
{michael.fellows,frances.rosamond}@cdu.edu.au

[3] The University of New South Wales and NICTA, Sydney, Australia
sergeg@cse.unsw.edu.au

Abstract. We introduce a method of applying Myhill-Nerode methods from formal language theory to hypergraphs and show how this method can be used to obtain the following parameterized complexity results.

– Hypergraph Cutwidth (deciding whether a hypergraph on n vertices has cutwidth at most k) is linear-time solvable for constant k.
– For hypergraphs of constant *incidence treewidth* (treewidth of the incidence graph), Hypertree Width and variants cannot be solved by simple finite tree automata. The proof leads us to conjecture that Hypertree Width is W[1]-hard for this parameter.

1 Introduction

This work extends the graph-theoretic analog [8] of the Myhill-Nerode characterization of regular languages to colored graphs and hypergraphs. Thus, we provide a method to derive linear-time algorithms (or to obtain evidence for intractability) for hypergraph problems on instances with bounded *incidence treewidth* (treewidth of the incidence graph). From a parameterized complexity point of view [7], incidence treewidth is an interesting parameter, since it can be bounded from above by canonical hypergraph width measures, like the treewidth of the primal graph [14] and the treewidth of the dual graph [18].

Applying Myhill-Nerode methods to hypergraphs, we obtain various parameterized complexity results, which we summarize in the following. Besides these results for hypergraph problems, our extension of the Myhill-Nerode theorem to colored graphs likely applies to other problems, since colored or annotated graphs allow for more realism in problem modeling and often arise as subproblems when solving pure graph problems. It is also straightforward to use our methods for annotated hypergraphs.

Hypergraph Cutwidth. We first apply our Myhill-Nerode approach to Hypergraph Cutwidth (see Section 3 for a formal definition)—a natural generalization of the NP-complete [11] and fixed-parameter tractable [7] Graph Cutwidth problem, for which several fixed-parameter algorithms are known

[1, 4, 9, 10, 19]. Cahoon and Sahni [5] designed algorithms for HYPERGRAPH CUTWIDTH with $k \leq 2$, with running time $O(n)$ for $k = 1$ and running time $O(n^3)$ for $k = 2$, where n is the number of vertices. For arbitrary k, Miller and Sudborough [16] designed an algorithm with running time $O(n^{k^2+3k+3})$. We suspect that the framework of Nagamochi [17] applies to HYPERGRAPH CUTWIDTH, giving an $n^{O(k)}$ time algorithm. The algorithm we present here has running time $O(n + m)$ for constant k, thus showing HYPERGRAPH CUTWIDTH to be *fixed-parameter linear* for the parameter k.

In the context of VLSI design, the HYPERGRAPH CUTWIDTH problem is known as BOARD PERMUTATION, and it is related to the gate matrix layout problem and several graph problems; see [16] and references therein.

Hypertree Width. The original Myhill-Nerode theorem can be used both positively and negatively: to show that a language is regular, and to show that a language is not regular. Using our hypergraph Myhill-Nerode analog negatively, we obtain evidence that the problems HYPERTREE WIDTH, GENERALIZED HYPERTREE WIDTH, and FRACTIONAL HYPERTREE WIDTH are not *fixed-parameter tractable* with respect to the parameter incidence treewidth t, that is, we conjecture that there are no algorithms for these problems running in time $f(t) \cdot n^c$, where n is the input size, c is a constant, and f is a computable function. It is already known that these problems are unlikely to be fixed-parameter tractable for their standard parameterizations [12, 13, 15]. Our result hints that even if the incidence width is constant, these other width measures cannot be computed efficiently.

Due to space constraints, we defer the proofs to a full version of this article [2].

Preliminaries. We use the standard graph-theoretic notions of Diestel [6].

Graph Decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i : i \in I\}, T)$ where $X_i \subseteq V$, $i \in I$, are called *bags* and T is a tree with elements of I as nodes such that:

– for each edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$, and
– for each vertex $v \in V$, $T[\{i \in I : v \in X_i\}]$ is a non-empty connected tree.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* of G is the minimum width taken over all tree decompositions of G. The notions of *path decomposition* and *pathwidth* of G are defined the same way, except that T is restricted to be a path.

Hypergraphs. A *hypergraph* H is a pair (V, E), where V is a set of *vertices* and E a multiset of *hyperedges* such that $e \subseteq V$ for each $e \in E$. Let $H = (V, E)$ be a hypergraph. The *primal graph* of H, denoted $\mathcal{G}(H)$, is the graph with vertex set V that has an edge $\{u, v\}$ if there exists a hyperedge in H incident to both u and v. It is sometimes called the Gaifman graph of H. The *incidence graph* of H, denoted $\mathcal{I}(H)$, is the bipartite graph (V', E') with vertex set $V' = V \cup E$ and for $v \in V$ and $e \in E$, there is an edge $\{v, e\} \in E'$ if $v \in e$.

Hypergraph Decompositions. Let H be a hypergraph. Generalized hypertree width is defined with respect to tree decompositions of $\mathcal{G}(H)$, however, the width of the tree decompositions is measured differently. Suppose H has no isolated vertices (otherwise, remove them). A *cover* of a bag is a set of hyperedges such that each vertex in the bag is contained in at least one of these hyperedges. The *cover width* of a bag is the minimum number of hyperedges covering it. The *cover width* of a tree decomposition is the maximum cover width of any bag in the decomposition. The *generalized hypertree width* of H is the minimum cover width over all tree decompositions of $\mathcal{G}(H)$.

The *hypertree width* of H is defined in a similar way, except that, additionally, the tree of the decomposition is rooted and a hyperedge e can only be used in the cover of a bag X_i if X_i contains all vertices of e that occur in bags of the subtree rooted at the node i.

The *fractional hypertree width* of H is defined in a similar way as the generalized hypertree width, except that it uses fractional covers: in a *fractional cover* of a bag, each hyperedge is assigned a non-negative weight, and for each vertex in the bag, the sum of the weights of the hyperedges incident to it is at least 1. The *fractional cover width* of the bag is the minimum total sum of all hyperedges of a fractional cover.

2 Myhill-Nerode for Colored Graphs and Hypergraphs

The aim of this section is first to generalize the Myhill-Nerode analog for graphs [8] to colored graphs. From this, we obtain a Myhill-Nerode analog for hypergraphs, since every hypergraph can be represented as its incidence graph with two vertex types: those representing hyperedges and those representing hypergraph-vertices.

In the last part of the section, we finally describe how our Myhill-Nerode analog yields linear-time algorithms for hypergraph problems. We follow and adapt the notation used by Downey and Fellows [7, Section 6.4].

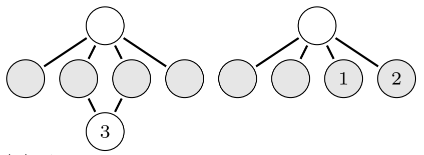
2.1 Colored Graphs

We now develop an analog of the Myhill-Nerode theorem for colored graphs. The original Myhill-Nerode theorem is stated for languages in terms of concatenations of words. Hence, we clarify what concatenating colored graphs means.

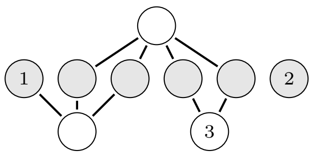
Definition 1. *A t-*boundaried graph *G is a graph with t distinguished vertices that are labeled from 1 to t. These labeled vertices are called* boundary vertices. *The* boundary, *$\partial(G)$, denotes the set of boundary vertices of G.*

Let G_1 and G_2 be t-boundaried graphs whose vertices are colored with colors from $\{1, \ldots, c_{\max}\}$. We say that G_1 and G_2 are color-compatible *if the vertices with the same labels in $\partial(G_1)$ and $\partial(G_2)$ have the same color.*

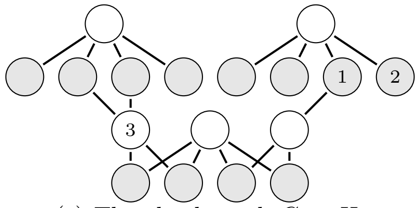
For two color-compatible t-boundaried graphs, we denote by $G_1 \oplus_c G_2$ the colored graph obtained by taking the disjoint union of G_1 and G_2 and identifying



(a) A 2-colored 3-boundaried graph G.



(b) A 2-colored 3-boundaried graph H.



(c) The glued graph $G \oplus_{\mathrm{c}} H$.

Fig. 1. Two color-compatible 3-boundaried graphs G and H and their glued graph, where the boundary vertices are marked by their label

each vertex of $\partial(G_1)$ with the vertex of $\partial(G_2)$ with the same label, wherein vertex colors are inherited from G_1 and G_2.

Let $\mathcal{U}^{large}_{t,c_{\max}}$ be the universe of $\{1,\ldots,c_{\max}\}$-colored t-boundaried graphs and $F \subseteq \mathcal{U}^{large}_{t,c_{\max}}$. We define the canonical right congruence $\sim_F$ *for F as follows: for $G_1, G_2 \in \mathcal{U}^{large}_{t,c_{\max}}$, $G_1 \sim_F G_2$ if and only if G_1 and G_2 are color-compatible and for all color-compatible $H \in \mathcal{U}^{large}_{t,c_{\max}}$, $G_1 \oplus_{\mathrm{c}} H \in F \iff G_2 \oplus_{\mathrm{c}} H \in F$.*

The index *of $\sim_F$ is its number of equivalence classes.*

Definition 1 is illustrated in Figure 1. Before we can state our analog of the Myhill-Nerode theorem for colored graphs, we show that every $\{1,\ldots,c_{\max}\}$-colored graph of treewidth at most t can be generated using a constant number of graph operations. To this end, we use the following set of operators. For generating graphs of only one color, the given operators coincide with those given by Downey and Fellows [7, Section 6.4].

Definition 2. *The* size-$(t+1)$ parsing operators *for $\{1,\ldots,c_{\max}\}$-colored graphs are:*

i) $\{\emptyset_{n_1,\ldots,n_{c_{\max}}} : \sum_{i=1}^{c_{\max}} n_i = t+1\}$ *is a family of nullary operators that creates boundary vertices $1,\ldots,t+1$, of which the first n_1 vertices get color 1, the next n_2 vertices get color 2, and so on.*

ii) γ *is a unary operator that cyclically shifts the boundary. That is, γ moves label j to the vertex with label $j+1 \pmod{t+1}$.*

iii) i is a unary operator that assigns the label 1 to the vertex currently labeled 2 and label 2 to the vertex with label 1.

iv) e is a unary operator that adds an edge between the vertices labeled 1 and 2.

v) $\{u_\ell : 1 \le \ell \le c_{\max}\}$ *is a family of unary operators that add a new vertex of color ℓ and label it 1, unlabeling the vertex previously labeled 1.*

vi) $\oplus_{\mathrm{c}}$ *is our gluing operator from Definition 1.*

For a constant number of colors $c_{\max}$, the set of size-$(t+1)$ parsing operators is finite. Adapting the proof of Downey and Fellows [7, Theorem 6.72], we verify that the graphs generated by the operators in Definition 2 have treewidth at most t. The same proof shows how, from a width-t tree decomposition of a colored graph G with at least $t+1$ vertices, a linear-size parse tree over the above operators can be obtained in linear time that generates a graph isomorphic to G.

Definition 3. *The set $\mathcal{U}^{small}_{t,c_{\max}}$ is the set of $\{1,\dots,c_{\max}\}$-colored t-boundaried graphs that can be generated by the operators in Definition 2.*

Theorem 1. *Let $F \subseteq \mathcal{U}^{small}_{t,c_{\max}}$ be a family of graphs. The following statements are equivalent:*

i) The parse trees corresponding to graphs in F are recognizable by a finite tree automaton.
ii) The canonical right congruence $\sim_F$ has finite index over $\mathcal{U}^{small}_{t,c_{\max}}$.

2.2 Lifting to Hypergraphs

To make the method accessible to hypergraph problems, we lift the Myhill-Nerode theorem for colored graphs of the previous subsection to hypergraphs.

Definition 4. *A t-boundaried hypergraph G has t distinguished vertices and hyperedges labeled from* 1 *to t. Two t-boundaried hypergraphs are* gluable *if no vertex of one hypergraph has the label of a hyperedge of the other hypergraph.*

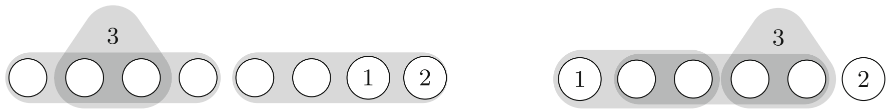
Let G_1 and G_2 be gluable t-boundaried hypergraphs. We denote by $G_1 \oplus_{\mathrm{h}} G_2$ the t-boundaried hypergraph obtained by taking the disjoint union of G_1 and G_2, identifying each labeled vertex of G_1 with the vertex of G_2 with the same label, and replacing the hyperedges with label ℓ by the union of these hyperedges.

Let $\mathcal{H}^{large}_t$ be the universe of t-boundaried hypergraphs and $F \subseteq \mathcal{H}^{large}_t$. We define the canonical right congruence *$\sim_F$ for F as follows: for $G_1, G_2 \in \mathcal{H}^{large}_t$, $G_1 \sim_F G_2$ if and only if G_1 and G_2 are gluable and for all $H \in \mathcal{H}^{large}_t$ that are gluable to G_1 and G_2, $G_1 \oplus_{\mathrm{h}} H \in F \iff G_2 \oplus_{\mathrm{h}} H \in F$.*

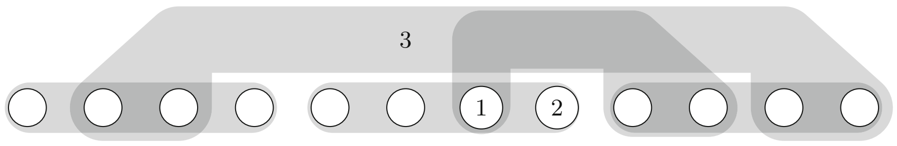
To prove a Myhill-Nerode theorem for hypergraphs, we still need a way to create hypergraphs from parsing operators, as we did using the operators from Definition 2 for colored graphs. To this end, we indeed simply use the operators from Definition 2, observing that every bipartite graph can be interpreted as the incidence graph of a hypergraph. Moreover, if the two disjoint independent sets of a two-colored bipartite graph have distinct colors, then we can interpret this bipartite graph as a hypergraph in a unique way.

Definition 5. *A* well-colored t-boundaried graph *is a $\{1,2\}$-colored t-boundaried graph $G = (U \uplus W, E)$, where the vertices in U have color 1, the vertices in W have color 2, and where U and W are independent sets.*

For a well-colored t-boundaried graph $G = (U \uplus W, E)$, we denote by $\mathcal{H}(G)$ the t-boundaried hypergraph with the vertex set U and the edge set $\{N(w) : w \in W\}$.



(a) A 3-boundaried hypergraph $\mathcal{H}(G)$. (b) A 3-boundaried hypergraph $\mathcal{H}(H)$.



(c) The glued hypergraph $\mathcal{H}(G) \oplus_{\mathrm{h}} \mathcal{H}(H) = \mathcal{H}(G \oplus_{\mathrm{c}} H)$.

Fig. 2. The two hypergraphs represented by the well-colored 3-boundaried graphs G and H in Figure 1 and the glued hypergraph $\mathcal{H}(G) \oplus_{\mathrm{h}} \mathcal{H}(H) = \mathcal{H}(G \oplus_{\mathrm{c}} H)$

Moreover, vertices of $\mathcal{H}(G)$ inherit their label from G and edges $e = N(w), w \in W$ of $\mathcal{H}(G)$ inherit the label of w. For a set $U \subseteq \mathcal{U}_{t,2}^{large}$, we denote $\mathcal{H}(U) := \{H \in \mathcal{H}_t^{large} \mid H = \mathcal{H}(G), G \in U\}$.

The incidence graph *of $\mathcal{H}(G)$ is G and the* incidence treewidth *of $\mathcal{H}(G)$ is the treewidth of G.*

Obviously, every $H \in \mathcal{H}_t^{\text{large}}$ is $\mathcal{H}(G)$ for some $G \in \mathcal{U}_{t,2}^{\text{large}}$. Also, every set $\mathcal{F} \subseteq \mathcal{H}_t^{\text{large}}$ is $\mathcal{F} = \mathcal{H}(F)$ for some $F \subseteq \mathcal{U}_{t,2}^{\text{large}}$. Figure 2 illustrates Definitions 4 and 5.

Theorem 2. *Let $\mathcal{F} \subseteq \mathcal{U}_{t,2}^{small}$ be a set of well-colored t-boundaried graphs. The following statements are equivalent:*

i) The parse trees corresponding to graphs in F are recognizable by a finite tree automaton.
ii) The canonical right congruence $\sim_{\mathcal{H}(F)}$ has finite index over $\mathcal{H}(\mathcal{U}_{t,2}^{small})$.

We can use Theorem 2 to constructively derive algorithms for deciding properties of hypergraphs of incidence treewidth at most t: assume that we have a family of t-boundaried hypergraphs $\mathcal{F} \subseteq \mathcal{H}_t^{\text{large}}$ with incidence treewidth at most t such that $\sim_{\mathcal{F}}$ has finite index over $\mathcal{H}_t^{\text{large}}$. There is a set $F \subseteq \mathcal{U}_{t,2}^{\text{large}}$ for which $\mathcal{F} = \mathcal{H}(F)$.

We can decide in linear time whether a given hypergraph $H \in \mathcal{H}_t^{\text{large}}$ is isomorphic to a hypergraph in $\mathcal{F}$: compute the incidence graph G of H, that is, $\mathcal{H}(G) = H$. Since the graph G has treewidth at most t, we can compute a tree decomposition for G in linear time [3]. In the same way as shown by Downey and Fellows [7, Theorem 6.72], this tree decomposition can be converted in linear time into a parse tree T over the operators in Definition 2 that generates a graph isomorphic to G. By Theorem 2, a finite tree automaton can check whether T generates a graph G' isomorphic to a graph in F, which is the case if and only if $\mathcal{H}(G')$ is isomorphic to some graph in $\mathcal{F}$. The finite tree automaton can be constructed from the equivalence classes of $\sim_{\mathcal{F}}$ in constant time, since each equivalence class has a constant-size representative.

3 Hypergraph Cutwidth is Fixed-Parameter Tractable

In this section we show that Hypergraph Cutwidth is fixed-parameter linear. We first formally define the problem.

Let $H = (V, E)$ be a hypergraph. A *linear layout* of H is an injective map $l\colon V \to \mathbb{R}$ of vertices onto the real line. The *cut at position* $i \in \mathbb{R}$ with respect to l, denoted $\theta_{l,H}(i)$, is the set of hyperedges that contain at least two vertices v, w such that $l(v) < i < l(w)$. The *cutwidth* of the layout l is $\max_{i \in \mathbb{R}} |\theta_{l,H}(i)|$. The *cutwidth* of the hypergraph H is the minimum cutwidth over all the linear layouts of H. The Hypergraph Cutwidth problem is then defined as follows.

Hypergraph Cutwidth
Input: A hypergraph $H = (V, E)$ and a natural number k.
Question: Does H have cutwidth at most k?

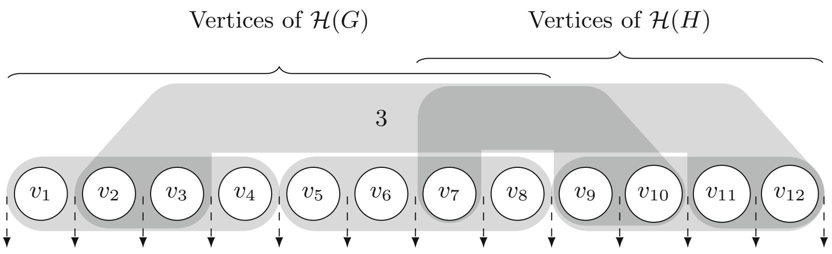
Now, to solve Hypergraph Cutwidth using our Myhill-Nerode analog, in the remainder of this section we consider a constant k and the class k-HCW of all hypergraphs with cutwidth at most k. We use Theorem 2 to show that the parse trees for graphs in k-HCW can be recognized by a finite tree automaton. To make Theorem 2 applicable, we first show that, for the hypergraphs in k-HCW, we can find a constant upper bound t on their incidence treewidth. This implies that isomorphic graphs can be generated by linear-size parse trees over the operator set in Definition 2 or, in terms of Theorem 2, that the graphs in k-HCW are isomorphic to the graphs in a subset of $\mathcal{H}(\mathcal{U}_{t,2}^{\text{small}})$.

Lemma 1. *Let $H = (V, E)$ be a hypergraph. If H has cutwidth at most k, then H has incidence treewidth at most* $\max\{k, 1\}$.

It remains to prove that the canonical right congruence $\sim_{k\text{-HCW}}$ of k-HCW has finite index. This finally shows that k-HCW can be recognized in linear time and, therefore, that Hypergraph Cutwidth is fixed-parameter linear.

To show that $\sim_{k\text{-HCW}}$ has finite index, we show that, given a t-boundaried hypergraph G, only a finite number of bits of information about a t-boundaried hypergraph H is needed in order to decide whether $G \oplus_{\text{h}} H \in k$-HCW. To this end, we employ the Method of Test Sets [7]: let $\mathcal{T}$ be a set of objects called *tests* (for the moment, it is not important what exactly a test is). A t-boundaried graph can *pass* a test. For t-boundaried hypergraphs G_1 and G_2, let $G_1 \sim_{\mathcal{T}} G_2$ if and only if G_1 and G_2 pass the same subset of tests in $\mathcal{T}$. Obviously, $\sim_{\mathcal{T}}$ is an equivalence relation. Our aim is to find a set $\mathcal{T}$ of tests such that $\sim_{\mathcal{T}}$ refines $\sim_{k\text{-HCW}}$ (that is, if $G_1 \sim_{\mathcal{T}} G_2$, then $G_1 \sim_{k\text{-HCW}} G_2$). This implies that if $\sim_{\mathcal{T}}$ has finite index, so has $\sim_{k\text{-HCW}}$. To show that $\sim_{\mathcal{T}}$ has finite index, we show that we can find a *finite* set $\mathcal{T}$ such that $\sim_{\mathcal{T}}$ refines $\sim_{k\text{-HCW}}$.

Intuitively, in our case, a hypergraph G will pass a test T if it has a restricted linear layout, where each of its boundary vertices gets mapped to predefined integer values and each of the remaining vertices “lands” within one of a set of given “landing zones” between the integer values of the real line. This restricted linear layout will impose the same restrictions on an optimal cutwidth layout



	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
w_i	0	0	0	0	0	0	0	1	1	2	1	2	0
E_i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{3\}$	$\emptyset$

Fig. 3. Construction of the H-test illustrated using the glued hypergraph $\mathcal{H}(G) \oplus_{\mathrm{h}} \mathcal{H}(H)$ from Figure 2

for G that are also imposed by an optimal cutwidth layout of $G \oplus_{\mathrm{h}} H$ for some hypergraph H that corresponds to T.

Definition 6. *A* landing zone *is a tuple in* $\{0, \ldots, k\} \times 2^{\{1,\ldots,t\}}$. *A* test $T = (\pi, S)$ *consists of a map* $\pi \colon \{1, \ldots, t\} \to \{1, \ldots, n\}$ *and a sequence* $S = (S_0, S_1, \ldots, S_n)$ *of landing zones. The* size *of* T *is* n.

Now, let G *and* H *be* t*-boundaried hypergraphs such that* $G \oplus_{\mathrm{h}} H \in k$-HCW. *Let* n *be the number of vertices of* $G \oplus_{\mathrm{h}} H$ *and let* v_i *denote the vertex that is mapped to position* i *in an optimal cutwidth layout of* $G \oplus_{\mathrm{h}} H$ *that maps to integer values. We define an* H-test $T = (\pi, S)$ *of size* n *as follows: for each vertex* $v_i \in \partial(H)$, *set* $\pi(\ell) := i$, *where* ℓ *is the label of* v_i. *For* $i \in \{0, \ldots, n\}$, $S_i := (w_i, E_i)$, *where*

1. w_i *is the number of hyperedges in* H *that contain vertices in* $\{v_1, \ldots, v_i\} \cap V(H)$ *and* $\{v_{i+1}, \ldots, v_n\} \cap V(H)$.
2. E_i *is the set of labels of hyperedges in* H *containing vertices in* $\{v_1, \ldots, v_i\} \cap V(H)$ *and* $\{v_{i+1}, \ldots, v_n\} \cap V(H)$.

Figure 3 illustrates this definition. We now formally define what it means to pass a test. Intuitively, if a graph G passes an H-test, then $G \oplus_{\mathrm{h}} H \in k$-HCW.

Definition 7. *Let* $G = (V, E)$ *be a* t*-boundaried hypergraph and* $T = (\pi, S)$ *be an* H*-test for some* t*-boundaried hypergraph* H, *where* $S = (S_0, \ldots, S_n)$ *and* $S_i = (w_i, E_i)$.

A T-compatible layout for G *is an injective function* $f \colon V \to \mathbb{R}$ *such that each vertex* $v \in \partial(G)$ *with label* ℓ *is mapped to* $\pi(\ell)$ *and such that every vertex* $v \in V \setminus \partial(G)$ *is mapped into some open interval* $(i, i+1)$ *for* $0 \leq i \leq n$.

The weighted cutwidth of f *is* $\max_{i \in \mathbb{R}}(|\theta_f(i)| + w_{\lfloor i \rfloor})$, *where* $\theta_f(i)$ *is the set of hyperedges containing two vertices* v *and* w *with* $f(v) < i < f(w)$ *and that do not have a label in* $E_{\lfloor i \rfloor}$.

Finally, G passes the test T *if there is a* T*-compatible layout* f *for* G *whose weighted cutwidth is at most* k.

Lemma 2. *For $\mathcal{T}$ being the set of all tests, the equivalence relation $\sim_{\mathcal{T}}$ refines $\sim_{k\text{-HCW}}$.*

Proof (Sketch). We show that if two t-boundaried hypergraphs G_1, G_2 pass the same subset of tests of $\mathcal{T}$, then, for all t-boundaried hypergraphs H, $G_1 \oplus_{\mathrm{h}} H \in k$-HCW if and only if $G_2 \oplus_{\mathrm{h}} H \in k$-HCW. We exploit the following two claims.

1. If $G_1 \oplus_{\mathrm{h}} H \in k$-HCW, then G_1 passes the H-test.
2. If G_2 passes the H-test, then $G_2 \oplus_{\mathrm{h}} H \in k$-HCW.

Let H be a t-boundaried hypergraph such that $G_1 \oplus_{\mathrm{h}} H \in k$-HCW, and let T be an H-test. By (1), G_1 passes T. Since G_1 and G_2 pass the same tests, also G_2 passes T. By (2), it follows that $G_2 \oplus_{\mathrm{h}} H \in k$-HCW. The reverse direction is proved symmetrically. □

Lemma 2 shows a set of tests $\mathcal{T}$ such that $\sim_{\mathcal{T}}$ refines $\sim_{k\text{-HCW}}$. However, the set $\mathcal{T}$ is infinite and, therefore, does not yet yield that $\sim_{k\text{-HCW}}$ has finite index. However, we can obtain a finite set of tests using the following lemma.

Lemma 3. *Let G and H be t-boundaried hypergraphs. For every H-test T_1, there is a test T_2 of size $2t(2k+2)$ such that G passes T_1 if and only if it passes T_2.*

Now the following theorem is easy to prove.

Theorem 3. Hypergraph Cutwidth *is fixed-parameter linear.*

Proof. Lemma 1 shows that graphs in k-HCW have constant treewidth at most t, and therefore, that all such hypergraphs can be linear-time transformed into linear-size parse trees for t-boundaried hypergraphs. Using Theorem 2, we show that parse trees corresponding to hypergraphs in k-HCW are recognizable by a finite tree automaton: let $\mathcal{T}$ be the set of all tests and $\mathcal{T}'$ be the set of all tests of size $2t(2k+2)$. Lemma 3 shows that $\sim_{\mathcal{T}'}$ refines $\sim_{\mathcal{T}}$. Lemma 2 shows that $\sim_{\mathcal{T}}$ refines $\sim_{k\text{-HCW}}$. Therefore, $\sim_{k\text{-HCW}}$ has at most as many equivalence classes as $\sim_{\mathcal{T}'}$. Since k and t are constant, $\mathcal{T}'$ is finite, implying finite index for $\sim_{\mathcal{T}'}$ and, consequently, for $\sim_{k\text{-HCW}}$. □

4 Hypertree Width and Variants

In this section we sketch a negative application of our hypergraph Myhill-Nerode analog to Generalized Hypertree Width [13]. The problem is, given a hypergraph H and an integer k as input, to decide whether H has generalized hypertree width at most k. Since Generalized Hypertree Width is NP-hard for $k = 3$ [13], it would be nice to find parameters for which the problem is fixed-parameter tractable. However, we can show that Generalized Hypertree Width does not have finite index.

Theorem 4. *Let $k \geq 0$. Let k-GHTW be the family of all incidence graphs G such that $\mathcal{H}(G)$ has generalized hypertree width at most k. The canonical right congruence $\sim_{\mathcal{H}(k\text{-GHTW})}$ does not have finite index over $\mathcal{H}(\mathcal{U}_{t,2}^{small})$.*

Moreover, the construction we use in the proof leads us to conjecture the problem to be W[1]-hard with respect to the parameter incidence treewidth. The proof also applies to the problem variants Hypertree Width and Fractional Hypertree Width, which are unlikely to be fixed-parameter tractable with respect to their standard parameterization [12, 15].

Conjecture 1. Generalized Hypertree Width is W[1]-hard with respect to the parameter incidence treewidth.

5 Conclusion

We have extended the graph analog of the Myhill-Nerode theorem to colored graphs and hypergraphs, making the methodology more widely applicable. Our positive application shows that Hypergraph Cutwidth is fixed-parameter linear. As a negative application, we showed that Hypertree Width, Generalized Hypertree Width, and Fractional Hypertree Width do not have finite index, and therefore the parse trees associated to Yes-instances of bounded incidence treewidth cannot be recognized by finite tree automata.

Acknowledgments. The authors are thankful to Mahdi Parsa for fruitful discussions. René van Bevern acknowledges support by the DFG, project DAPA (NI 369/12-1). The other authors acknowledge support by by the Australian Research Council, grants DP 1097129 (Michael R. Fellows), DE 120101761 (Serge Gaspers), and DP 110101792 (Michael R. Fellows and Frances A. Rosamond). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] Abrahamson, K.R., Fellows, M.R.: Finite automata, bounded treewidth, and well-quasiordering. In: Graph Structure Theory. Contemporary Mathematics, vol. 147, pp. 539–564. American Mathematical Society (1991)
[2] van Bevern, R., Fellows, M.R., Gaspers, S., Rosamond, F.A.: Myhill-nerode methods for hypergraphs (2013), arxiv:1211.1299v3 (cs.DM)
[3] Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
[4] Bodlaender, H.L., Fellows, M.R., Thilikos, D.M.: Derivation of algorithms for cutwidth and related graph layout parameters. J. Comput. Syst. Sci. 75(4), 231–244 (2009)
[5] Cahoon, J., Sahni, S.: Exact algorithms for special cases of the board permutation problem. In: Proceedings of the 21st Annual Allerton Conference on Communication, Control, and Computing, pp. 246–255 (1983)
[6] Diestel, R.: Graph Theory, 4th edn. Graduate Texts in Mathematics, vol. 173. Springer, New York (2010)
[7] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)

[8] Fellows, M.R., Langston, M.A.: An analogue of the Myhill-Nerode Theorem and its use in computing finite-basis characterizations (extended abstract). In: Proc. 30th FOCS, pp. 520–525. IEEE Computer Society (1989)
[9] Fellows, M.R., Langston, M.A.: On well-partial-order theory and its application to combinatorial problems of VLSI design. SIAM J. Discrete Math. 5(1), 117–126 (1992)
[10] Fellows, M.R., Langston, M.A.: On search, decision, and the efficiency of polynomial-time algorithms. J. Comput. Syst. Sci. 49(3), 769–779 (1994)
[11] Gavril, F.: Some NP-complete problems on graphs. In: Proc. 1977 Conf. on Inf. Sc. and Syst., pp. 91–95. Johns Hopkins Univ. (1977)
[12] Gottlob, G., Grohe, M., Musliu, N., Samer, M., Scarcello, F.: Hypertree decompositions: Structure, algorithms, and applications. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 1–15. Springer, Heidelberg (2005)
[13] Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. J. ACM 56(6) (2009)
[14] Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. J. Comput. Sci. 61(2), 302–332 (2000)
[15] Marx, D.: Approximating fractional hypertree width. ACM Transactions on Algorithms 6(2), 1–29 (2010)
[16] Miller, Z., Sudborough, I.H.: A polynomial algorithm for recognizing bounded cutwidth in hypergraphs. Mathematical Systems Theory 24(1), 11–40 (1991)
[17] Nagamochi, H.: Linear layouts in submodular systems. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 475–484. Springer, Heidelberg (2012)
[18] Samer, M., Szeider, S.: Constraint satisfaction with bounded treewidth revisited. J. Comput. Sci. 76(2), 103–114 (2010)
[19] Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Cutwidth I: A linear time fixed parameter algorithm. J. Algorithms 56(1), 1–24 (2005)

Augmenting Graphs to Minimize the Diameter

Fabrizio Frati[1], Serge Gaspers[2,3],
Joachim Gudmundsson[1,3], and Luke Mathieson[4]

[1] University of Sydney, Australia
brillo@it.usyd.edu.au, joachim.gudmundsson@sydney.edu.au
[2] The University of New South Wales, Australia
sergeg@cse.unsw.edu.au
[3] NICTA, Australia
[4] Macquarie University, Australia
luke.mathieson@mq.edu.au

Abstract. We study the problem of augmenting a weighted graph by inserting edges of bounded total cost while minimizing the diameter of the augmented graph. Our main result is an FPT 4-approximation algorithm for the problem.

1 Introduction

We study the problem of minimizing the diameter of a weighted graph by the insertion of edges of bounded total cost. This problem arises in practical applications [2,4] such as telecommunications networks, information networks, flight scheduling, protein interactions, and it has also received considerable attention from the graph theory community, see for example [1,7,11].

We introduce some terminology. Let $G = (V, E)$ be an undirected weighted graph. Let $[V]^2$ be the set of all possible edges on the vertex set V. A *non-edge* of G is an element of $[V]^2 \setminus E$. The *weight* of a path in G is the sum of its edge weights. For any $u, v \in V$, the *shortest u-v path* in G is the path connecting u and v in G with minimum weight. The weight of this path is said to be the *distance* between u and v in G. Finally, the *diameter* of G is the largest distance between any two vertices in G. The problem we study in this paper is formally defined as follows.

PROBLEM:	Bounded Cost Minimum Diameter Edge Addition (BCMD)
INPUT:	An undirected graph $G = (V, E)$, a weight function $w : [V]^2 \to \mathbb{N}$, a cost function $c : [V]^2 \to \mathbb{N}^*$, and an integer B.
GOAL:	A set F of non-edges with $\sum_{e \in F} c(e) \leq B$ such that the diameter of the graph $G_B = (V, E \cup F)$ with weight function w is minimized. We say that G_B is a *B-augmentation* of G.

The main result of this paper is a fixed parameter tractable (FPT) 4-approximation algorithm for BCMD with parameter B. FPT approximation algorithms are surveyed by Marx [14]. For background on parameterized complexity we refer to [6,8,15] and for background on approximation algorithms to [17].

Several papers in the literature already dealt with the BCMD problem. However, most of them focused on restricted versions of the problem, namely the one in which all costs and all weights are identical [3,5,12,13], and the one in which all the edges have unit costs and the weights of the non-edges are all identical [2,4].

The BCMD problem can be seen as a bicriteria optimization problem where the two optimization criteria are: (1) the cost of the edges added to the graph and (2) the diameter of the augmented graph. As is standard in the literature, we say that an algorithm is an (α, β)-approximation algorithm for the BCMD problem, with $\alpha, \beta \geq 1$, if it computes a set F of non-edges of G of total cost at most $\alpha \cdot B$ such that the diameter of $G' = (V, E \cup F)$ is at most $\beta \cdot D^B_{opt}$, where D^B_{opt} is the diameter of an optimal B-augmentation of G.

We survey some known results about the BCMD problem. Note that all the algorithms discussed below run in polynomial time.

Unit Weights and Unit Costs. The restriction of BCMD to unit costs and unit weights was first shown to be NP-hard in 1987 by Schoone et al. [16]; see also the paper by Li et al. [13]. Bilò et al. [2] showed that, as a consequence of the results in [3,5,13], there exists no $(c \log n, \delta < 1 + 1/D^B_{opt})$-approximation algorithm for BCMD if $D^B_{opt} \geq 2$, unless P=NP. For the case in which $D^B_{opt} \geq 6$, they proved a stronger lower bound, namely that there exists no $(c \log n, \delta < \frac{5}{3} - \frac{7-(D^B_{opt}+1) \mod 3}{3D^B_{opt}})$-approximation algorithm, unless P=NP.

Dodis and Khanna [5] gave an $(O(\log n), 2+2/D^B_{opt})$-approximation algorithm (see also [12]). Li et al. [13] showed a $(1, 4+2/D^B_{opt})$-approximation algorithm. The analysis of the latter algorithm was later improved by Bilò et al. [2], who showed that it gives a $(1, 2 + 2/D^B_{opt})$-approximation. In the same paper they also gave a $(O(\log n), 1)$-approximation algorithm.

Unit Costs and Restricted Weights. Some of the results from the unweighted setting have been extended to a restricted version of the weighted case, namely the one in which the edges of G have arbitrary non-negative integer weights, however all the non-edges of G have cost 1 and uniform weight $\omega \geq 0$.

Bilò et al. [2] showed how two of their algorithms can be adapted to this restricted weighted case. In fact, they gave a $(1, 2 + 2\omega/D^B_{opt})$-approximation algorithm and a $(2 - 1/B, 2)$-approximation algorithm. Similar results were obtained by Demaine and Zadimoghaddam in [4].

Bilò et al. [2] also showed that, for every $D^B_{opt} \geq 2\omega$ and for some constant c, there is no $(c \log n, \delta < 2 - 3\omega/D^B_{opt})$-approximation algorithm for this restriction of the BCMD problem, unless P=NP.

Arbitrary Costs and Weights. To the best of our knowledge, only one theory paper considered the general BCMD problem. In 1999, Dodis and Khanna [5] presented an $O(n \log D^B_{opt}, 1)$-approximation algorithm, assuming that all weights are polynomially bounded. Their result is based on a multicommodity flow formulation of the problem.

Our Results. In this paper we study the BCMD problem with arbitrary integer costs and weights. Our main result is a $(1, 4)$-approximation algorithm with running time $O((3^B B^3 + n + \log(Bn))Bn^2)$. We also prove that, considering B as a parameter, it is $W[2]$-hard to compute a $(1 + c/B, 3/2 - \epsilon)$-approximation, for any constants c and $\epsilon > 0$. Further, we present polynomial-time $((k + 1)^2, 3)$-, $(k, 4)$-, and $(1, 3k + 2)$-approximation algorithms for the unit-cost restriction of the BCMD problem.

2 Shortest Paths with Bounded Cost

Let $(G = (V, E), w, c, B)$ be an instance of the BCMD problem and let K denote the complete graph on the vertex set V. The edges of K have the same weights and costs as they have in G (observe that an edge e of K is either an edge or a non-edge of G). For technical reasons, we add self-loops with weight 0 and cost 1 at each vertex of K.

For any $0 \le \beta \le B$, a path in K is said to be a *β-bounded-cost path* if it uses non-edges of G of total cost at most β. We consider the problem of computing, for *every* integer $0 \le \beta \le B$ and for every two vertices $u, v \in V$, a β-bounded-cost shortest path connecting u and v, if such a path exists. We call this problem the *All-Pairs B-Shortest Paths* (APSP$_B$) problem. We will prove the following.

Theorem 1. *The APSP$_B$ problem can be solved in $O(Bn^3 + Bn^2 \log(Bn))$ time using $O(Bn^2)$ space.*

In order to prove Theorem 1, we construct a directed graph $H = (U, F)$ as follows. First, consider G as a directed graph, i.e., replace every undirected edge $\{u, v\}$ with two arcs (u, v) and (v, u) with the same weight and cost as the edge $\{u, v\}$. Then, $H = (U, F)$ contains $B + 1$ copies of G, denoted by $G_0, \ldots, G_B$. For any $0 \le i \le B$, we denote by (v, i) the copy of vertex $v \in V$ in $G_i = (V_i, E_i)$. The arc set F contains the union of E' and F', where $E' = \bigcup_{0 \le i \le k} E_i$, and

$$F' = \Big\{ \big((u, i), (v, i + c(\{u, v\}))\big) \;:\; 0 \le i \le B - c(\{u, v\}),\ \{u, v\} \in [V]^2 \setminus E \Big\}.$$

For each $((u, i), (v, j)) \in F'$, the weight and the cost of $((u, i), (v, j))$ are $w(\{u, v\})$ and $c(\{u, v\}) = j - i$, respectively.

Observation 1 *The number of vertices in U is $(B + 1)n$ and the number of arcs in F is $O(Bn^2)$.*

We will use directed graph H to efficiently compute β-bounded-cost shortest paths in K. This is possible due to the following two lemmata.

Lemma 1. *Suppose that H contains a directed path P_H with weight W connecting vertices (u, i) and (v, j), for some $j \ge i$. Then, there exists a $(j - i)$-bounded-cost path P_K in K with weight W connecting u and v.*

Lemma 2. *Suppose that there exists a β-bounded-cost path P_K in K with weight W connecting vertices u and v. Then, there exists a directed path P_H in H with weight W connecting vertices $(u, 0)$ and (v, β).*

We have the following.

Corollary 1. *There is a β-bounded-cost path connecting vertices u and v in K with weight W if and only if there is a directed path in H connecting vertices $(u, 0)$ and (v, β) with weight W.*

We are now ready to prove Theorem 1. Consider any vertex u in K. We first mark every vertex that can be reached from $(u, 0)$ in H with the weight of its shortest path from $(u, 0)$. By Observation 1, H has $O(Bn)$ vertices and $O(Bn^2)$ edges, hence this can be done in $O(Bn^2 + Bn \log(Bn))$ time [9]. For every $0 \leq \beta \leq B$ and for every vertex $v \neq u$, by Corollary 1 the weight of a β-bounded cost shortest path in K is the same as the weight of a shortest directed path from $(u, 0)$ to (v, β) in H. Hence, for every $0 \leq \beta \leq B$ and for every vertex $v \neq u$, we can determine in total $O(Bn^2 + Bn \log(Bn))$ time the weight of a β-bounded cost shortest path in K connecting u and v. Thus, for every $0 \leq \beta \leq B$ and for every pair of vertices u and v in K, we can determine in total $O(Bn^3 + Bn^2 \log(Bn))$ time the weight of a β-bounded cost shortest path in K connecting u and v. This concludes the proof of Theorem 1.

3 Arbitrary Costs and Weights

Our algorithms, as many afore-mentioned approximation algorithms for the BCMD problem, use a clustering approach as a first phase to find a set C of $B+1$ cluster centers. The idea of the algorithm is to create a minimum height rooted tree $T = (U \subseteq V, D)$, so that $C \subseteq U$, by adding a set of edges of total cost at most B to G. We will prove that such a tree approximates an optimal B-augmentation.

3.1 Clustering

We start by defining the clustering approach used to generate the $B + 1$ cluster centers. Whereas a costly binary search is used in [4] to guess the radius of the clusters, we adapt the approach of [2] to our more general setting.

For two vertices u, v, we denote by $\text{dist}_G(u, v)$ the distance between u and v in G. For a vertex u and a set of vertices S, we denote by $\text{dist}_G(u, S)$ the minimum distance between u and any vertex from S in G, i.e., $\text{dist}_G(u, S) = \min_{v \in S}\{\text{dist}_G(u, v)\}$. For a set of vertices S, we denote by $\text{dist}_G(S)$ the minimum distance between any two distinct vertices from S in G, i.e., $\text{dist}_G(S) = \min_{u \in S}\{\text{dist}_G(u, S \setminus \{u\})\}$.

The clustering phase computes a set $C = \{c_1, \dots, c_{B+1}\}$ of $B + 1$ cluster centers as follows. Vertex c_1 is an arbitrary vertex in V; for $2 \leq i \leq B + 1$, vertex c_i is chosen so that $\text{dist}_G(c_i, \{c_1, \dots, c_{i-1}\})$ is maximized. Ties are broken arbitrarily.

Lemma 3. *The clustering phase computes in $O(Bn^2)$ time a set $C \subseteq V$ of size $B + 1$ such that $\text{dist}_G(v, C) \leq D^B_{opt}$ for every vertex $v \in V$.*

Proof. First, note that the above described algorithm can easily be implemented in $O(Bn^2)$ time using B iterations of Dijkstra's algorithm with Fibonacci heaps [9]. Let c_{B+2} denote a vertex maximizing $\text{dist}_G(c_{B+2}, C)$, and denote this distance by R. By definition, $\text{dist}_G(v, C) \leq R$ for every $v \in V$. To prove the lemma it remains to show that $R \leq D^B_{opt}$. For the sake of contradiction, assume $D^B_{opt} < R$. Then, $C \cup \{c_{B+2}\}$ is a set of $B + 2$ vertices with pairwise distance larger than D^B_{opt} in G. We prove the following claim.

Claim 1. *Let G' be a weighted graph and let C' be a set of vertices in G' such that $\text{dist}_{G'}(C') > D$. Then, for every graph G'' obtained from G' by adding a single non-edge of G' with non-negative weight, there is a set $C'' \subset C'$ with $|C''| = |C'| - 1$ and with $\text{dist}_{G''}(C'') > D$.*

Now, since $C \cup \{c_{B+2}\}$ is a set of $B + 2$ vertices with pairwise distance larger than D^B_{opt} in G, by iteratively using the claim we have that in any B-augmentation G_B of

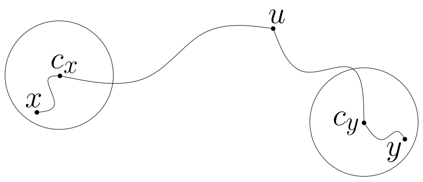


Fig. 1. Illustrating the path defined in the proof of Lemma 5

G, we have a set of $B + 2 - |F| \geq 2$ vertices with pairwise distance greater than D^B_{opt}, thus contradicting the definition of D^B_{opt}. This concludes the proof of the lemma. □

3.2 A Minimum Height Tree

Let C be a set of $B + 1$ cluster centers such that the $B + 1$ clusters with centers at $C = \{c_0, \ldots, c_B\}$ and radius D^B_{opt} cover the vertices of G. This set can be computed as described in the previous section.

Definition 1. *Let $G = (V, E)$ be a graph together with a weight function $w : [V]^2 \to \mathbb{N}$. Let $C \subseteq V$ and let u be a vertex in V. A Shortest Path Tree of G, C, and u, denoted by* $\text{SPT}(G, C, u)$*, is a tree T rooted at u, spanning C, whose vertices and edges belong to V and E, respectively, and such that, for every vertex $c \in C$, it holds $d_T(u, c) = d_G(u, c)$.*

The *height* of a weighted rooted tree T, which is denoted by $\hbar(T)$, is the maximum weight of a path from the root to a leaf.

Definition 2. *Let $G = (V, E)$ be a graph together with a weight function $w : [V]^2 \to \mathbb{N}$ and a cost function $c : [V]^2 \to \mathbb{N}^*$. Let $C \subseteq V$, let u be a vertex in V, and let $B \geq 0$ be an integer. A Minimum Height$_B$ SPT of G, C, and u, denoted by* $\text{MH}_B\text{SPT}\ (G, c, u)$*, is a* $\text{SPT}(G_B, C, u)$ *of minimum height over all B-augmentations G_B of G.*

Let G_B be a B-augmentation of G with diameter D^B_{opt}.

Lemma 4. *The height of a* $\text{MH}_B\text{SPT}\ (G, C, u)$ *is at most D^B_{opt}.*

Proof. By definition, we have (A) $\hbar(\text{MH}_B\text{SPT}(G, C, u)) \leq \hbar(\text{SPT}(G_B, C, u))$. Since G_B is a B-augmentation of G with diameter D^B_{opt}, we have (B) $\hbar(\text{SPT}(G_B, C, u)) \leq D^B_{opt}$. Inequalities (A) and (B) together prove the lemma. □

We now present a relationship between the BCMD problem and the problem of computing a $\text{MH}_B\text{SPT}\ (G, C, u)$.

Lemma 5. *Let G'_B be a B-augmentation of G such that it holds $\hbar(\text{SPT}(G'_B, C, u)) = \hbar(\text{MH}_B\text{SPT}(G, C, u))$, for any $u \in V$. Then, the diameter of G'_B is at most $4 \cdot D^B_{opt}$.*

Proof. Consider two vertices x and y in V, see Fig. 1. Let c_x and c_y be centers of the clusters x and y belong to, respectively. Then, we have $\text{dist}_{G'_B}(x, y) \leq \text{dist}_G(x, c_x) + \text{dist}_{G'_B}(c_x, u) + \text{dist}_{G'_B}(u, c_y) + \text{dist}_G(c_y, y)$. By Lemma 3, $\text{dist}_G(x, c_x)$, $\text{dist}_G(c_y, y) \leq D^B_{opt}$. Since $\hbar(\text{SPT}(G'_B, C, u)) = \hbar(\text{MH}_B\text{SPT}(G, C, u))$ and by Lemma 4, it holds $\text{dist}_{G'_B}(c_x, u), \text{dist}_{G'_B}(u, c_y) \leq D^B_{opt}$. Hence, $\text{dist}_{G'_B}(x, y) \leq 4 \cdot D^B_{opt}$. □

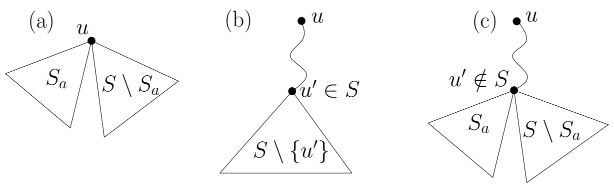


Fig. 2. Illustration for the proof of Lemma 6

3.3 Constructing a Minimum Height Tree

In this section, we show an algorithm to compute a MH$_B$SPT (G, C, c_1).

We introduce some notation and terminology. Let $C' = C \setminus \{c_1\}$. Observe that a MH$_B$SPT (G, C', c_1) is also a MH$_B$SPT (G, C, c_1), given that a MH$_B$SPT (G, C', c_1) contains c_1 as its root. Denote by $d_K^j(u, v)$ the minimum weight of a j-bounded cost path connecting u and v in K. For any $u \in V$, for any $S \subseteq C'$, and for any $0 \leq j \leq B$, let $\gamma(u, S, j)$ denote the height of a MH$_j$SPT (G, S, u). Hence, the height of a MH$_B$SPT (G, C', c_1) is $\gamma(c_1, C', B)$. The following main lemma gives a dynamic programming recurrence for computing $\gamma(c_1, C', B)$.

Lemma 6. *For any $u \in V$, any $S \subseteq C'$, and any $0 \leq j \leq B$, the following hold: If $|S| = 1$, then $\gamma(u, S, j) = d_K^j(u, c_i)$ where $S = \{c_i\}$. If $|S| > 1$, then*

$$\gamma(u, S, j) = \min_{\substack{v \in V \\ S' \subsetneq S \\ j = j_1 + j_2 + j_3}} d_K^{j_1}(u, v) + \max\{\gamma(v, S', j_2), \gamma(v, S \setminus S', j_3)\}.$$

Proof. If $|S| = \{c_i\}$, then MH$_j$SPT $(G, \{c_i\}, u)$ is a minimum-weight path connecting u and c_i and having total cost at most j. Hence, $\gamma(u, S, j) = d_K^j(u, c_i)$. In particular, notice that, if $u = c_i$, then $\gamma(u, \{u\}, j) = d_K^j(u, u) = 0$.

If $|S| = m > 1$, then suppose that the lemma holds for each $\gamma(u', S', j')$ with $|S'| \leq m - 1$ by induction. Denote by T any MH$_j$SPT (G, S, u). Denote by $P(v, w)$ the unique path in T connecting two vertices v and w of T. We distinguish three cases, based on the structure of T. In Case (a), the degree of u in T is at least two (see Fig. 2(a)). In Case (b), the degree of u in T is one and there exists a vertex $u' \in S$ such that every internal vertex of $P(u, u')$ has degree 2 in T and does not belong to S (see Fig. 2(b)). Finally, in Case (c), the degree of u in T is one and there exists a vertex $u' \notin S$ such that every internal vertex of $P(u, u')$ has degree 2 in T and does not belong to S, and such that the degree of u' is greater than two (see Fig. 2(c)).

First, we prove that one of the three cases always applies. If the degree of u in T is at least two, then Case (a) applies. Otherwise, the degree of u is 1. Traverse T from u until a vertex v' is found such that $v' \in S$ or the degree of v' is at least 3. If $v' \in S$, then every internal vertex of $P(u, u')$ has degree 2 in T and does not belong to S, hence Case (b) applies. If $v' \notin S$, then the degree of v' is at least 3, and every internal vertex of $P(u,u')$ has degree 2 in T and does not belong to S, hence Case (c) applies. We now discuss the three cases.

In Case (a), T is composed of two subtrees MH_xSPT (G, S_a, u) and MH_ySPT $(G, S \setminus S_a, u)$, only sharing vertex u, with $\emptyset \subsetneq S_a \subsetneq S$. The height of T is the maximum of the heights of MH_xSPT (G, S_a, u) and MH_ySPT $(G, S \setminus S_a, u)$; also the cost of T is at most $x + y$. By induction, the heights of MH_xSPT (G, S_a, u) and MH_ySPT $(G, S \setminus S_a, u)$ are $\gamma(u, S_a, x)$ and $\gamma(u, S \setminus S_a, y)$), respectively. Thus, the height of T is $\max\{\gamma(v, S_a, x), \gamma(v, S \setminus S_a, y)\}$ and hence $\gamma(u, S, j) = \max\{\gamma(u, S_a, x), \gamma(u, S \setminus S_a, y)\}$. Such a value is found by the recursive definition of $\gamma(u, S, j)$ with $v = u$, $S' = S_a$, $j_1 = 0$, $j_2 = x$, and $j_3 = y$.

In Case (b), T is composed of a path from u to u' with cost x and weight $d^x_K(u,u')$, and of a MH_ySPT $(G, S \setminus \{u'\}, u')$. The height of T is the sum of $d^x_K(u,u')$ and the height of MH_ySPT $(G, S \setminus \{u'\}, u')$; also the cost of T is at most $x + y$. By induction, the height of MH_ySPT $(G, S \setminus \{u'\}, u')$ is $\gamma(u', S \setminus \{u'\}, y)$. Thus, the height of T is $d^x_K(u,u') + \gamma(u', S \setminus \{u'\}, y)$ and hence $\gamma(u, S, j) = d^x_K(u,u') + \gamma(u', S \setminus \{u'\}, y)$. Such a value is found by the recursive definition of $\gamma(u, S, j)$ with $v = u'$, $S' = S \setminus \{u'\}$, $j_1 = x$, $j_2 = y$, and $j_3 = 0$.

In Case (c), T is composed of a path from u to u' with cost x and weight $d^x_K(u,u')$, of a MH_ySPT (G, S_a, u'), and of a MH_zSPT $(G, S \setminus S_a, u')$ with $\emptyset \subsetneq S_a \subsetneq S$. The height of T is the sum of $d^x_K(u,u')$ and the maximum between the heights of MH_ySPT (G, S_a, u') and MH_zSPT $(G, S \setminus S_a, u')$; also the cost of T is at most $x + y + z$. By induction, the heights of MH_ySPT (G, S_a, u') and MH_zSPT $(G, S \setminus S_a, u')$ are $\gamma(u', S_a, y)$ and $\gamma(u', S \setminus S_a, z)$, respectively. Thus, the height of T is $d^x_K(u,u') + \max\{\gamma(u', S_a, y), \gamma(u', S \setminus S_a, z)\}$ and hence $\gamma(u, S, j) = d^x_K(u,u') + \max\{\gamma(u', S_a, y), \gamma(u', S \setminus S_a, z)\}$. Such a value is found by the recursive definition of $\gamma(u, S, j)$ with $v = u'$, $S' = S_a$, $j_1 = x$, $j_2 = y$, and $j_3 = z$.

This concludes the induction and hence the proof of the lemma. □

Lemma 6 yields the following.

Theorem 2. *There exists a* $(1, 4)$*-approximation algorithm for the* BCMD *problem with* $O((3^B B^3 + n + \log(Bn))Bn^2)$ *running time.*

4 Unit Costs and Arbitrary Weights

For the special case in which each edge has unit cost and arbitrary weight, our techniques lead to several results, that are described in the following. Observe that, in this case we are allowed to insert in G exactly k non-edges of G, where $k = B = O(n^2)$. We remark that Theorem 2 gives a $(1, 4)$-approximation algorithm running in $O((3^k k^3 + n)kn^2)$ time for this special case.

In the following, we denote by C a clustering with $k + 1$ clusters constructed as described in Subsection 3.1. We first show a $((k + 1)^2, 3)$-approximation algorithm.

Theorem 3. *Given an instance of the* BCMD *problem with unit costs, there exists a* $((k + 1)^2, 3)$*-approximation algorithm with* $O(kn^3)$ *running time.*

Proof. For every pair of cluster centers $c_i, c_j \in C$ compute a shortest path in K between c_i and c_j that contains at most k non-edges of G. Add those edges to F and let

$G' = (V, E \cup F)$. By Theorem 1 and since $k = O(n^2)$, G' can be constructed in $O(kn^3)$ time. Observe that, for each pair of cluster centers, the algorithm adds at most k non-edges of G to F, thus at most $k(k+1)^2$ non-edges in total. We prove that, for every $v_i, v_j \in V$, there exists a path in G' connecting v_i and v_j whose weight is at most $3 \cdot D^k_{opt}$. Denote by c_i and c_j the centers of the clusters v_i and v_j belong to, respectively. We have $\text{dist}_{G'}(v_i, v_j) \leq \text{dist}_{G'}(v_i, c_i) + \text{dist}_{G'}(c_i, c_j) + \text{dist}_{G'}(c_j, v_j)$. By Lemma 3, $\text{dist}_{G'}(v_i, c_i), \text{dist}_{G'}(v_j, c_j) \leq D^k_{opt}$; also, by construction, $\text{dist}_{G'}(c_i, c_j) \leq D^k_{opt}$, and the theorem follows. □

Next, we give a $(k, 4)$-approximation algorithm.

Theorem 4. *Given an instance of the* BCMD *problem with unit costs, there exists a $(k, 4)$-approximation algorithm with $O(kn^2)$ running time.*

Proof. Pick an arbitrary cluster center, say c_1. For every cluster center $c_j \in C \setminus \{c_1\}$, compute a shortest path between c_1 and c_j in K containing at most k non-edges of G. Add those edges to F and let $G' = (V, E \cup F)$. By Corollary 1, a shortest path between c_1 and c_j in K containing at most k non-edges of G corresponds to a shortest path between $(c_1, 0)$ and (c_j, k) in digraph H. By Observation 1, H has $O(kn)$ vertices and $O(kn^2)$ edges. Hence, Dijkstra's algorithm with Fibonacci heaps [9] computes all the shortest paths between $(c_1, 0)$ and (c_j, k), for every $c_j \in C \setminus \{c_1\}$, in total $O(kn^2)$ time. Observe that, for each cluster different from c_1, the algorithm adds at most k non-edges of G to F, thus at most k^2 non-edges in total. We prove that, for every $v_i, v_j \in V$, there exists a path in G' connecting v_i and v_j whose weight is at most $4 \cdot D^k_{opt}$. Denote by c_i and c_j the centers of the clusters v_i and v_j belong to, respectively. We have $\text{dist}_{G'}(v_i, v_j) \leq \text{dist}_{G'}(v_i, c_i) + \text{dist}_{G'}(c_i, c_1) + \text{dist}_{G'}(c_1, c_j) + \text{dist}_{G'}(c_j, v_j)$. By Lemma 3, $\text{dist}_{G'}(v_i, c_i), \text{dist}_{G'}(v_j, c_j) \leq D^k_{opt}$; by construction, $\text{dist}_{G'}(c_i, c_1), \text{dist}_{G'}(c_1, c_j) \leq D^k_{opt}$, and the theorem follows. □

Finally, we present a $(1, 3k + 2)$-approximation algorithm.

Theorem 5. *Given an instance of the* BCMD *problem with unit costs, there exists a $(1, 3k + 2)$-approximation algorithm with $O(n^2 + k^2)$ running time.*

Proof. For every pair of clusters C_i and C_j, with $1 \leq i < j \leq k + 1$, let e_{ij} be the edge of minimum weight connecting a vertex in C_i with a vertex in C_j. We denote by F' the set of these edges. For a subset F of F', we say that F *spans* C if the graph representing the adjacencies between clusters via the edges of F is connected. Let F be a minimum-weight set of k edges from F' spanning C. Let $G' = (V, E \cup F)$. The set F', and hence the graph G', can be constructed in $O(n^2 + k^2)$ time as follows. Consider all the edges of K and keep, for each pair of clusters, the edge with smallest weight. This can be done in $O(n^2)$ time. Finally, compute in $O(k^2)$ time a minimum spanning tree of the resulting graph [10], that has $O(k)$ vertices and $O(k^2)$ edges. Observe that the algorithm adds at most k non-edges of G to F. We prove that, for every $v_i, v_j \in V$, there exists a path in G' connecting v_i and v_j whose weight is at most $(3k + 2)D^k_{opt}$. Denote by P_C the (unique) subset of F connecting the clusters v_i and v_j belong to. Let $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ be the edges of P_C in order from v_i to v_j. Then, $\text{dist}_{G'}(v_i, v_j) \leq \text{dist}_G(v_i, x_1) + w(x_1, y_1) +$

$\text{dist}_G(y_1, x_2) + \ldots + w(x_m, y_m) + \text{dist}_G(y_m, v_j)$. By Lemma 3, $\text{dist}_G(y_i, x_{i+1}) \leq 2D^k_{opt}$, and $\text{dist}_G(v_i, x_1), \text{dist}_G(y_m, v_j) \leq D^k_{opt}$. Also, $w((x_i, y_i)) \leq D^k_{opt}$, and the theorem follows. □

5 Hardness Results

The main theorem of this section provides a parameterized intractability result for BCMD with unit weights and unit costs, and some related problems. The U-BCMD problem has as input an unweighted graph $G = (V, E)$ and two integers k and d, and the question is whether there is a set $F \subseteq [V]^2 \setminus E$, with $|F| \leq k$, such that the graph $(V, E \cup F)$ has diameter at most d. The parameter is k. We will show that U-BCMD is $W[2]$-hard. We will also provide refinements to the minimum conditions required for intractability, namely U-BCMD remains NP-complete for graphs of diameter 3 with target diameter $d = 2$. We note that although Dodis and Kanna [5] provide an inapproximability reduction from SET COVER, they begin with a disconnected graph, and expand the instance with a series of size 2 sets, which does not preserve the size of the optimal solution, and therefore their reduction cannot be used to show parameterized complexity lower bounds.

Theorem 6. SET COVER *is polynomial-time reducible to* U-BCMD. *Moreover, the reduction is parameter preserving and creates an instance with diameter* 3 *and target diameter* 2.

Proof. Let (X, S, k) be an instance of SET COVER where S is the base set and $X \subset \mathcal{P}(S)$ is the set from which we must pick the set cover of S with size at most k. We construct an instance $(G = (V, E), k, d)$ of U-BCMD as follows.

Let $m = |X| \cdot k$. The vertex set V is the disjoint union of 5 sets:

– a set Y corresponding to the set X where for each $x \in X$ we have a vertex $y \in Y$,
– a set $T = \biguplus_{i \in [m]} T_i$ corresponding to S where, for each $s \in S$ and $i \in [m]$, we have a vertex $t_i \in T_i$ (*i.e.,* we have m copies of a set of vertices corresponding to S),
– a set U with $\binom{m}{2}$ vertices u_{ij}, one for each pair of disjoint subsets T_i, T_j of T (where $i \neq j$),
– the set $\{a\}$, and
– the set $\{b\}$.

The edge set E consists of the following edges:

– ab,
– by for each vertex $y \in Y$,
– bu_{ij} for each vertex $u_{ij} \in U$,
– yy' for each pair of vertices $y, y' \in Y$,
– yt_i for each pair of vertices $y \in Y$ and $t_i \in T_i$ for each $i \in [m]$ where the corresponding element $s \in S$ is in the corresponding set $x \in X$ in the SET COVER instance,
– $t_i u_{jl}$ for each pair of vertices $t_i \in T_i$ and $u_{jl} \in U$ such that $i \in \{j, l\}$, and
– $u_{ij} u_{lp}$ for each pair of vertices $u_{ij}, u_{lp} \in U$.

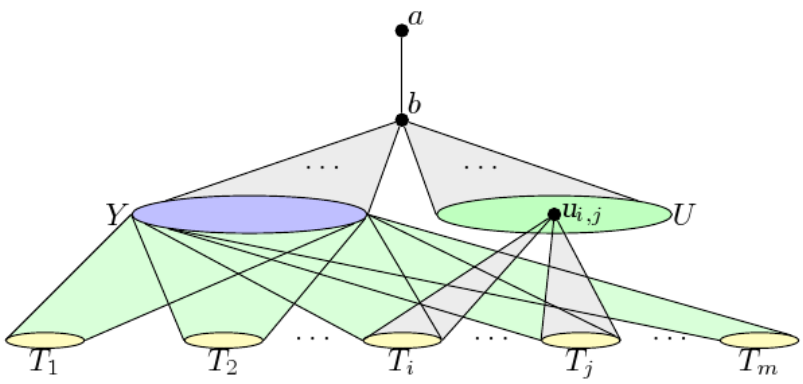


Fig. 3. Sketch of the construction for the SET COVER to U-BCMD reduction. The edge sets represented in gray are complete, the edge sets represented in light green correspond to the set membership from the SET COVER instance. The vertex sets Y and U are cliques. The vertex sets T_i are independent sets for all $i \in [m]$.

We set $d = 2$. Note that k in the U-BCMD instance is the same k as for the SET COVER instance. The construction is sketched in Fig. 3.

Claim 2. *For all* $v, v' \in V \setminus \{a\}$ *we have* $\mathrm{dist}(v, v') \leq 2$.

Claim 3. *For all* $v \in V$ *we have* $\mathrm{dist}(a, v) \leq 3$. *Moreover,* $\mathrm{dist}(a, v) = 3$ *if and only if* $v \in T$.

Thus we are concerned only with reducing the distance between a and the vertices of T.

Claim 4. (X, S, k) *is a* YES*-instance of* SET COVER *if and only if* (G, k, d) *is a* YES*-instance of* U-BCMD.

We note that the reduction is obviously polynomial-time computable, and the parameter k is preserved. The theorem now follows from the previous claims. □

Corollary 2. U-BCMD *is* NP*-complete even for graphs of diameter three with target diameter two.*

As SET COVER is $W[2]$-hard with parameter k, combined with Corollary 2 we also have the following result.

Corollary 3. U-BCMD *is* $W[2]$*-hard even for graphs of diameter three with target diameter two.*

We note additionally that as the initial graph has diameter 3 and the target diameter is 2, it is even NP-hard and W[2]-hard to decide if there is a set of k new edges that improves the diameter by one. Furthermore by taking a as source vertex, the results transfer immediately to the single-source version as discussed by Demaine & Zadimoghaddam [4].

The construction of Theorem 6 can even be extended to give a parameterized inapproximability result for U-BCMD.

Theorem 7. *It is $W[2]$-hard to compute a $(1+\frac{c}{k}, \frac{3}{2}-\varepsilon)$-approximation for* U-BCMD *for any constants c and $\varepsilon > 0$.*

Acknowledgments. SG acknowledges support from the Australian Research Council (grant DE120101761). JG is funded by the Australian Research Council FT100100755. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Alon, N., Gyárfás, A., Ruszinkó, M.: Decreasing the diameter of bounded degree graphs. Journal of Graph Theory 35, 161–172 (1999)
2. Bilò, D., Gualà, L., Proietti, G.: Improved approximability and non-approximability results for graph diameter decreasing problems. Theoretical Computer Science 417, 12–22 (2012)
3. Chepoi, V., Vaxès, Y.: Augmenting trees to meet biconnectivity and diameter constraints. Algorithmica 33(2), 243–262 (2002)
4. Demaine, E.D., Zadimoghaddam, M.: Minimizing the diameter of a network using shortcut edges. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 420–431. Springer, Heidelberg (2010)
5. Dodis, Y., Khanna, S.: Designing networks with bounded pairwise distance. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), pp. 750–759 (1999)
6. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
7. Erdős, P., Rényi, A., Sós, V.T.: On a problem of graph theory. Studia Sci. Math. Hungar. 1, 215–235 (1966)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory, Texts in Theoretical Computer Science. An EATCS Series, vol. XIV. Springer, Berlin (2006)
9. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (1987)
10. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. Journal of Computer and System Sciences 48(3), 533–551 (1994)
11. Grigorescu, E.: Decreasing the diameter of cycles. Journal of Graph Theory 43(4), 299–303 (2003)
12. Kapoor, S., Sarwat, M.: Bounded-diameter minimum-cost graph problems. Theory of Computing Systems 41(4), 779–794 (2007)
13. Li, C.-L., McCormick, S.T., Simchi-Levi, D.: On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. Operations Research Letters 11(5), 303–308 (1992)
14. Marx, D.: Parameterized complexity and approximation algorithms. The Computer Journal 51(1), 60–78 (2008)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford (2006)
16. Schoone, A.A., Bodlaender, H.L., van Leeuwen, J.: Diameter increase caused by edge deletion. Journal of Graph Theory 11, 409–427 (1997)
17. Vazirani, V.V.: Approximation algorithms. Springer (2001)

Top-k Document Retrieval in Compact Space and Near-Optimal Time

Gonzalo Navarro[1,⋆] and Sharma V. Thankachan[2,⋆⋆]

[1] Dept. of Computer Science, University of Chile, Chile
gnavarro@dcc.uchile.cl
[2] Dept. of Computer Science, Louisiana State University, USA
thanks@csc.lsu.edu

Abstract. Let $\mathcal{D}=\{d_1, d_2, ...d_D\}$ be a given set of D string documents of total length n. Our task is to index $\mathcal{D}$ such that the k most relevant documents for an online query pattern P of length p can be retrieved efficiently. There exist linear space data structures of $O(n)$ words for answering such queries in optimal $O(p+k)$ time. In this paper, we describe a compact index of size $|\mathsf{CSA}| + n \lg D + o(n \lg D)$ bits with near optimal time, $O(p+k \lg^* n)$, for the basic relevance metric *term-frequency*, where $|\mathsf{CSA}|$ is the size (in bits) of a compressed full-text index of $\mathcal{D}$, and $\lg^* n$ is the iterated logarithm of n.

1 Introduction and Related Work

Top-k document retrieval is the problem of preprocessing a text collection so that, given a search pattern $P[1..p]$ and a threshold k, we retrieve the k documents most "relevant" to P, for some definition of relevance. This is the basic problem of search engines and forms the core of the Information Retrieval (IR) field [5]. In this paper we focus on the popular *term frequency* as the relevance measure, that is, the number of times P appears in a document.

The inverted index successfully solves top-k queries in various IR scenarios. However, it applies to text collections that can be segmented into "words", so that only whole words can be queried. This excludes East Asian languages such as Chinese and Korean, where automatic segmenting is an open problem, and is troublesome even in languages such as German and Finnish. A simple solution for those cases is to treat the text as a plain sequence of symbols and look for any substring in those sequences. This string model is also appealing in applications like bioinformatics and software repositories. Supporting document retrieval on those general string collections has proved much more challenging.

Sufix trees [29] and arrays [15] are useful tools to search string collections. These structures solve the *pattern matching problem*, that is, count or list all the occ individual occurrences of P in the collection. Obtaining the k most relevant documents from that set requires time $\Omega(\mathsf{occ})$, usually much higher than k. Only recently [13,9,12,18,26] was the top-k problem solved satisfactorily,

⋆ Funded in part by Fondecyt Grant 1-110066.

⋆⋆ Supported by National Science Foundation (NSF) Grants CCF–1017623 (R. Shah and J. S. Vitter) and CCF–1218904 (R. Shah).

finally reaching the optimal time $O(p + k)$. Those solutions, like suffix trees, have the drawback of requiring $O(n \lg n)$ bits of space on a collection of length n, whereas the collection itself would require no more than $n \lg \sigma$ bits, where σ is the alphabet size. This renders these indexes impractical on large text collections.

Compressed Suffix Arrays (CSAs) satisfactorily solve the pattern matching problem within the size of the compressed text collection, under some entropy model [17]. They can in addition retrieve any substring of any document, and hence replace the collection with a compressed version that in addition supports queries. We call their space $|\mathsf{CSA}| \le n \lg \sigma (1 + o(1))$, which can be thought of as the minimum space in which the text collection can be represented.

A similar result for top-k queries has been sought. Various solutions use $2|\mathsf{CSA}| + o(n)$ bits [24,12,7,3], culminating with the fastest solution so far in this family, $O(p+k \lg k \lg^{1+\epsilon} n)$ time by Hon et al. [11]. Recently, asymptotically optimal space $|\mathsf{CSA}| + o(n)$ bits was obtained as well [27], being $O(p+k \lg^2 k \lg^{1+\epsilon} n)$ the best time achieved so far [20].

In all those solutions there is a significant time factor per element returned, of at least $\lg k \lg^{1+\epsilon} n$. It seems unlikely that this factor can disappear in this type of solutions. Experimental comparisons [6,19] show that these schemes are impractically slow compared to those that use $n \lg D + o(n \lg D)$ bits to store a so-called *document array* [16,28]. We call *compact* the solutions that use $|\mathsf{CSA}| + n \lg D + o(n \lg D)$ bits. The best practical results to date [6,21,3,14] are nearly compact. Their space requirement, 1–3 times the collection size (and including it), while not optimal, is affordable in many practical situations.

It is therefore relevant to ask which is the best time performance that can be achieved within compact space. The time-optimal result of Navarro and Nekrich [18], $O(p + k)$ time, requires $O(n(\lg D + \lg \sigma))$ bits. While of the same order of compact solutions, the constants are still way too large in practice. There have been some attempts to achieve truly compact solutions. Hon et al. [10] obtained $O(p + (\lg \lg n)^6 + k(\lg \sigma \lg \lg n)^{1+\epsilon})$ time, for any constant $\epsilon > 0$, using compact space. Alternatively, they obtain time $O(p + (\lg \lg n)^4 + k \lg \lg n)$ using $|\mathsf{CSA}| + 2n \lg D + o(n \lg D)$ bits. Konow and Navarro [14] achieved time $O(p + (\lg \lg n)^2 + k \lg \lg n)$ using $|\mathsf{CSA}| + n(\lg D + 4 \lg \lg n)(1 + o(1))$ bits, but the result holds only on typical texts, not in the worst case.

In this paper we show that it is possible to get very close to optimal time within compact space. We prove the following result, where we remark that the top-k results are not returned in sorted order of relevance.

Theorem 1. *There exists a compact index of* $|\mathsf{CSA}| + n \lg D + o(n \lg D)$ *bits and near-optimal* $O(p + k \lg^* n)$ *query time time, for the (unsorted) top-k frequent document retrieval problem, where* $\lg^* n$ *is the iterated logarithm of* n.

In Section 5 we achieve $O(p + k \lg^* k)$ time, using $o(n \lg \sigma)$ further bits.

2 The Data Structure

Three main components of our structure are a generalized suffix tree (GST), the document array, and some precomputed answer lists. These are described next.

Generalized Suffix Tree (GST): Let $T = d_1 d_2 d_3 ... d_D$ be the text (of length n) obtained by concatenating all the documents in $\mathcal{D}$. The last character of each document is \$, a special symbol that does not appear anywhere else in T. Each substring $T[i..n]$, with $i \in [1..n]$, is called a *suffix* of T. The *suffix tree* for T (or, equivalently, the *generalized suffix tree (GST)* of $\mathcal{D}$) is a lexicographic arrangement of all these n suffixes in a compact trie structure, where the ith leftmost leaf represents the ith lexicographically smallest suffix. Each edge in the suffix tree is labeled by a string, and $path(x)$ of a node x (node x refers to the node with preorder rank x) is the concatenation of edge labels along the path from the *root* of GST to node x. Let ℓ_i for $i \in [1..n]$ represent the (preorder rank of) the ith leftmost leaf in GST. Then $path(\ell_i)$ represents the ith lexicographically smallest suffix of T. A node x is called the *locus* of a pattern P, if it is the node closest to the *root* with $path(x)$ prefixed by P.

The *suffix array* $SA[1..n]$ is an array of length n, where $SA[i]$ is the starting position (in T) of the ith lexicographically smallest suffix of T. An important property of SA is that the starting positions of all the suffixes with the same prefix are always stored in a contiguous region of SA. Based on this property, we define the *suffix range* of P in SA to be the maximal range $[sp, ep]$ such that for all $i \in [sp, ep]$, $SA[i]$ is the starting point of a suffix of T prefixed by P.

A compressed representation of suffix array is called a Compressed Suffix Array (CSA). We will use a recent CSA [1], which obtains high-order entropy compression and can compute the suffix range $[sp, ep]$ of any given pattern $P[1..p]$ in $O(p)$ time. We also maintain the tree topology of GST in (at most) $4n + o(n)$ bits [25], with constant-time support of the operations *parent*(x) (the parent of node x), *lca*(x, y) (the lowest common ancestor of nodes x and y), *left-leaf*(x)/*right-leaf*(x) (the leftmost/rightmost leaf in the subtree rooted at node x), and *leaf*(i) (the ith leftmost leaf), and mapping from nodes to their preorder ranks and back. The total space of this component is $|\mathsf{CSA}| + O(n)$ bits.

Document Array (DA): Define a bit-vector $B[1..n]$, such that $B[i] = 1$ iff $T[i] = \$$. Then suffix $T[i, n]$ belongs to document d_r if $r = 1 + rank_B(i)$, where $rank_B(i)$ is the number of 1s in $B[1, i]$. The document array $DA[1..n]$ is defined as $DA[j] = r$ if the suffix $SA[j]$ belongs to document d_r. Moreover, we say that the corresponding leaf node ℓ_j is *marked* with document d_r. Now,

- $rank_{DA}(r, i)$ returns the number of occurrences of r in $DA[1, i]$;
- $select_{DA}(r, j)$ returns i where $DA[i] = r$ and $rank_{DA}(r, i) = j$; and
- $access_{DA}(i)$ returns $DA[i]$;

Then we have use the following representation for DA [2].

Lemma 1. *The document array DA can be stored in $n \lg D + o(n \lg D)$ bits and support queries $rank_{DA}$, $select_{DA}$ and $access_{DA}$ in times $O(\lg\lg n)$, $O(f(n, D))$ and $O(1)$ respectively, where $f(n, D) = \omega(1)$ is any non-constant function.*

The so-called *partial rank* query can be added to this repertoire [3].

Lemma 2. *Operation $rank_{DA}(DA[i], i)$ can be supported in constant time by storing $O(n \lg\lg D) = o(n \lg D)$ additional bits on top of the DA.*

Thus the total space of this component is $n \lg D + o(n \lg D)$ bits.

Precomputed Answer Lists: We start with the following definitions:

- $L(x)$ is the set of leaves in the subtree of node x in GST.
- $L(x \backslash y) = L(x) \setminus L(y)$, the leaves in the subtree of x, but not in that of y.
- $score(r, x)$ is the number of leaves in $L(x)$ marked with document d_r (i.e., $|\{\ell_i \in L(x), DA[i] = r\}|$).

We use the following scheme to identify a subset S_g of *marked nodes* in GST [12,21]: Let g be a parameter called *grouping factor*, then mark every gth leftmost leaf in GST, and then mark the lowest common ancestor (LCA) of every consecutive pair of marked leaves. Then, we have the following lemma [12,21].

Lemma 3. *The above marking scheme ensures the following properties:*

1. *The number of marked nodes is $|S_g| = \Theta(n/g)$.*
2. *If it exists, the closest marked descendant node y of any unmarked node x is unique, and $|L(x \backslash y)| < 2g$.*
3. *If there exists no marked node in the subtree of x, then $|L(x)| < 2g$.*

Let $F(x, k)$ represent the list (or set) of top-k documents d_r, along with $score(r, x)$, corresponding to a pattern with locus node x in GST. Clearly we cannot afford to maintain $F(x, k)$ for all possible x's and k's. Rather, we will maintain the lists $F(x, z)$ only for marked nodes x's (for various g values) and for k's that are powers of 2. Then $F(x, k)$ for any x and k will be efficiently computed using that sampled data. The next section describes how we store and retrieve the sampled lists.

3 Storing and Retrieving the Lists $F(x, z)$

The following is a key result in our scheme.

Lemma 4. *Let $g_h = z(\lg^{(h)} n)^2$ for any $1 \le h < \lg^* n$, where $\lg^{(1)} n = \lg n$, $\lg^{(h)} n = \lg(\lg^{(h-1)} n)$, and $\lg^{(\lg^* n)} n \le 1$. Then $F(x, z)$ for all $x \in S_{g_h}$ can be encoded in $s_h = s_{h-1} + O(n/\lg^{(h)} n)$ bits, and $F(x, z)$ for any given $x \in S_{g_h}$ can be decoded in time $t_h = t_{h-1} + O(z)$, where $s_1 = O(n/\lg n)$ and $t_1 = O(z)$.*

Proof. We use induction. Consider the base case $h = 1$. For every $x \in S_{g_1}$, we maintain the list $F(x, z)$ explicitly (using $O(\lg n)$ bits per element), along with a pointer to the location where it is stored, in $s_1 = O(|S_{g_1}| z \lg n) = O(n/\lg n)$ bits. Thus the list $F(x, z)$, for any $x \in S_{g_1}$, can be decoded in time $t_1 = O(z)$.

Now consider the grouping factor is g_h for $h \ge 2$. As we cannot afford to use $\Theta(\lg n)$ bits per element, we introduce encoding schemes that reduce it to $O(\lg^{(h)} n)$ bits. Thus the overall space for maintaining $F(x, z)$ (in encoded form) for all $x \in S_{g_h}$ can be bounded by $O(|S_{g_h}| z \lg^{(h)} n) = O(n/\lg^{(h)} n)$ bits. Instead of using pointers as in the base case, we mark in a bitmap $B^h[1..2n]$ the node preorders of GST that belong to S_{g_h}. Therefore the list $F(x,z)$ of a node $x \in S_{g_h}$ is stored in an array at offset $rank_{B^h}[x]$. Since we will only compute *rank* on positions x where $B^h[x] = 1$, an "indexed dictionary" is sufficient [23], which requires $O((n/g_h)\lg g_h + \lg\lg n) = o(n/\lg^{(h)} n)$ bits and computes *rank* in time $O(1)$. We now show how to encode the list $F(x,z)$, for $x \in S_{g_h}$, in $O(\lg^{(h)} n)$ bits per element, and how to decode it in $t_{h-1} + O(z)$ time.

We will maintain a structure STR_h, using s_h bits, for each grouping factor g_h, and will decode $F(x,z)$ for $x \in S_{g_h}$ recursively, using $O(z)$ time in addition to the time needed to decode $F(y,z)$ for some $y \in S_{g_{h-1}}$, as suggested in Lemma 4. As we cannot afford to sort the documents within the targeted query time, it is important to assume a fixed arrangement of documents within any particular decoded list $F(\cdot,\cdot)$. That is, each time we decode a specific list, the decoding algorithm must return the elements in the same order.

Let x be a node in S_{g_h} and y (if it exists) be its highest descendant node in $S_{g_{h-1}}$. We show how to encode and decode $F(x,z)$. To decode $F(x,z)$, we first decode the list $F(y,z)$ using STR_{h-1} in time t_{h-1}. From now onwards we have constant-time access to any element the list $F(y,z)$. The the list $F(x,z)$ will be partitioned into the following two disjoint lists:

(i) D_{old}, the documents that are common to $F(x,z)$ and $F(y,z)$.
(ii) D_{new}, the documents that are present in $F(x,z)$, but not in $F(y,z)$.

Encoding and decoding document identifers in D_{old}. We maintain a bit vector $B'[1..z]$, where $B'[i] = 1$ iff the ith document in $F(y,z)$ is present in $F(x,z)$. Therefore D_{old} can be decoded by listing those elements in $F(y,z)$ (in the same order as they appear) at positions i where $B'[i] = 1$. Thus space for maintaining the encoded information is z bits and the time for decoding is $O(z)$.

Encoding and decoding document identifers in D_{new}. For each document $d_r \in D_{new}$, there exists at least one leaf in $L(x\backslash y)$ that is marked with d_r (otherwise $score(r,x) = score(r,y)$ and d_r could not be in $F(x,z)$ and not in $F(y,z)$). Therefore, instead of explicitly storing r, it is sufficient to refer to such a leaf. For this we shall store a bit vector $B''[1..|L(x\backslash y)|]$ with all its bits in 0, except for $|D_{new}|$ 1's: for every document $d_r \in D_{new}$, we set one bit, say $B''[i] = 1$, where the ith leaf in $L(x\backslash y)$ is marked with d_r. Since $|B''| = |L(x\backslash y)| < 2g_{h-1}$ and the number of 1's is at most z, B'' can be encoded in $O(z\lg(g_{h-1}/z)) = O(z\lg^{(h)} n)$ bits with constant time *select* support [22] ($select_{B''}(j)$ is the position of the j-th 1 in B''). Now, given B'', the documents in D_{new} can be identified in $O(z)$ time as follows: Find all those (at most z) increasing positions i where $B''[i] = 1$ using *select* queries. Then, for each such i, find the ith leaf $\ell_{i'} \in L(x\backslash y)$ in constant time using the tree operations[1]. Finally, report $d_{DA[i']}$ as a document in D_{new} for each such i' using a constant-time *access* operation on the document array.

As mentioned before, it is important for our (recursive) encoding/decoding algorithm to assume a fixed permutation of elements within any list $F(\cdot,\cdot)$. We

[1] Compute the leftmost leaves ℓ_{i_x} and ℓ_{i_y}, respectively, of x and y, then $\ell_{i'}$ is ℓ_{i_x+i-1}, if $i_x + i - 1 < i_y$, and $\ell_{j_y+i-(i_y-i_x)}$ otherwise, where ℓ_{j_y} is the rightmost leaf of y.

use the convention that, in $F(x,z)$, the documents in D_{old} come before the documents in D_{new}. Moreover the documents within D_{old} and D_{new} are in the same order as the decoding algorithm identified them. In conclusion, the list of identifiers of documents in $F(x,z)$ can be encoded in $O(z \lg^{(h)} n)$ bits and decoded in $O(z)$ time, assuming constant-time access to any element in $F(y,z)$. If node y does not exist, we proceed as if $F(y,z) = \emptyset$ and $F(x,z) = D_{new}$. We now consider how to encode the *score*'s associated with the elements in $F(x,z)$ (i.e., $\mathit{score}(r,x)$ for all $d_r \in F(x,z)$).

Encoding and decoding of scores. Let d_{r_i}, for $i \in [1..z]$, be the ith document in $F(x,z)$, and $f_i = \mathit{score}(r_i,x)$. Then, define $\delta_i = f_i - f'_i \geq 0$, where

$$f'_i = \begin{cases} \mathit{score}(r_i,y) & \text{if } i \leq |D_{old}| \text{ (i.e., if } r_i \in D_{old}), \\ \tau = \min\{\mathit{score}(r,y), r \in F(y,z)\} & \text{if } i > |D_{old}| \text{ (i.e., if } r_i \in D_{new}). \end{cases}$$

The following is an important observation: The number of leaves in $L(x\backslash y)$ marked with document d_{r_i} is $\mathit{score}(r_i,x) - \mathit{score}(r_i,y)$, which is same as δ_i for $i \leq |D_{old}|$. For $i > |D_{old}|$, $\mathit{score}(r_i,x) - \mathit{score}(r_i,y) \geq \delta_i$, otherwise $\mathit{score}(r_i,y) > \tau$ and d_{r_i} would have qualified as a top-z document in $F(y,z)$ (which is a contradiction as $d_{r_i} \in D_{new}$). By combining with the fact that each leaf node is marked with a unique document, we have the inequality $\sum_{i=1}^{z} \delta_i \leq |L(x\backslash y)| < 2g_{h-1}$. Therefore, δ_i for all $i \in [1..z]$ can be encoded using a bit vector $B''' = 10^{\delta_1}10^{\delta_2}10^{\delta_3}\ldots10^{\delta_z}$ of length at most $2g_{h-1}+z$ with z 1's, in $O(z \lg(g_{h-1}/z)) = O(z \lg^{(h)} n)$ bits with constant-time *select* support [22].

The decoding algorithm is described as follows: compute the f'_i's for $i = 1\ldots z$ in the ascending order of i. For $i \leq |D_{old}|$, f'_i is given by score associated with the $(\mathit{select}_{B'}[i])$th document (which is same as d_{r_i}) in $F(y,z)$. This takes only $O(z)$ time as the number of constant-time *select* operations is $O(z)$, and we have constant-time access to any element and score in $F(y,z)$. Next, $\tau = \min\{\mathit{score}(r,y), r \in F(y,z)\}$ can be obtained by scanning the list $F(y,z)$ once. Thus all the f'_i's are computed in $O(z)$ time. Next we decode each δ_i and add it to f'_i to obtain f_i, for $i = 1\ldots z$ in $O(z)$ time, where $\delta_i = \mathit{select}_{B'''}(i) - \mathit{select}_{B'''}(i-1) - 1$ is computed in $O(1)$ time. Thus the space for maintaining the scores is $O(z \lg^{(h)} n)$ bits and the time for decoding them is $O(z)$.

Adding over the h levels, the total space is $s_h = s_{h-1} + O(n/\lg^{(h)} n) = O(n/\lg^{(h)} n)$ bits and the total decoding time is $t_h = t_{h-1} + O(z) = O(zh)$ (note that $s_1 = O(n/\lg n)$ and $t_1 = O(z)$). This completes the proof. □

4 Completing the Picture

Let $\pi \in [1..\lg^* n)$ be an integer such that $\lg^{(\pi-1)} n \geq \sqrt{\lg^* n} > \lg^{(\pi)} n$, then $\lg^{(\pi)} n = \omega(1)$ (note that $\pi = \lg^* n - \lg^* \sqrt{\lg^* n} = \Theta(\lg^* n)$). Then, by choosing g_π as the grouping factor, the space s_π is $O(n/\lg^{(\pi)} n) = o(n)$ bits. We maintain $\lg D$ such structures corresponding to $z = 1,2,4,8,...,2^{\lfloor \lg D \rfloor}$, in $o(n \lg D)$ bits total space. By combining the space bounds of all the components, we obtain the following lemma.

Lemma 5. *The total space requirement of our data structure is* $|\mathsf{CSA}|+n\lg D+o(n\lg D)$ *bits.*

The next lemma gives the total time to extract the sampled results and hints how we will use them.

Lemma 6. *Given any node* q *in* GST *and an integer* k*, our data structure can report the list* $F(q',k)$ *in* $O(k\lg^* n)$ *time, where* q' *is a node in the subtree of* q *with* $|L(q\backslash q')| = O(k\sqrt{\lg^* n})$*.*

Proof. As the first step, round k to $z = 2^{\lceil \lg k \rceil}$, which is the next highest power of 2. Then identify the highest node q', in the subtree of q, that is marked with respect to the grouping factor g_π: Let ℓ_i and ℓ_j be the leftmost and rightmost leaves of q in GST, then $q' = lca(\ell_{i'}, \ell_{j'})$ where $i' = g_\pi \cdot \lceil i/g_\pi \rceil$ and $j' = g_\pi \cdot \lfloor j/g_\pi \rfloor$ (there is no q' if $i' \geq j'$). This takes constant time on our representation of the GST topology.

Since $g_\pi = z\lg^{(\pi)} n < z\sqrt{\lg^* n}$, from Lemma 3 it holds $|L(q\backslash q')| = O(g_\pi) = O(z\lg^{(\pi)} n) = O(k\sqrt{\lg^* n})$. As $q' \in S_{g_\pi}$, the list $F(q',z)$ can be decoded in time $t_\pi = O(z\pi) = O(z\lg^* n)$ from the precomputed lists (from Lemma 4). The final $F(q',k)$ can be obtained by filtering those documents in $F(q',z)$ with score at least θ by a single scan of the list, where θ is the kth highest score in $F(q',z)$ (which can be computed in $O(z) = O(k)$ time using the linear-time selection algorithm [4]). In case q' does not exist, we report $F(q',k) = \emptyset$, and even in such a case the inequality condition $|L(q)| < 2g_\pi$ is guaranteed (from Lemma 3). □

4.1 Query Answering

The query answering algorithm consists of the following steps:

1. Find the locus node q of the input pattern P in GST by first obtaining the suffix range $[sp, ep]$ of P using CSA in $O(p)$ time, and then computing the lowest common ancestor of ℓ_{sp} and ℓ_{ep} in $O(1)$ time.
2. Using Lemma 6, find the node q' in the subtree of q, where $|L(q\backslash q')| = O(k\sqrt{\lg^* n})$ and retrieve the list $F(q',k)$ in $O(k\lg^* n)$ time.
3. Every document d_r in the final output $F(q,k)$ must either belong to $F(q',k)$, or it must be that $r = DA[i]$ for some leaf $\ell_i \in L(q\backslash q')$. Let us call S_{cand} the union of both sets of candidate documents. Then we compute $score(r,q)$ of each document $d_r \in S_{cand}$.
4. Report k documents in S_{cand} with the highest $score(r,q)$ value. In this step, we first compute the kth highest score θ using the selection algorithm, and then use θ as a threshold for a document to be an output (more precisely, we report the $k' < k$ documents $d_r \in S_{cand}$ with $score(r,q) < \theta$ in a first pass, and then report the first $k-k'$ documents $d_r \in S_{cand}$ we find with $score(r,q) = \theta$ in a second pass). The time is $O(|S_{cand}|) = O(k\sqrt{\lg^* n})$.

The overall time for Steps 1, 2, and 4 is $O(p + k\lg^* n)$. In the remaining part of this section we show how to handle Step 3 efficiently as well, for the documents

$r = DA[i]$ we find in $L(q \backslash q')$. Note that $score(r, q)$ can be computed as $rank_{DA}(r, ep) - rank_{DA}(r, sp-1)$ using two *rank* queries on the document array, but those *rank* queries are expensive. Instead, we use a more sophisticated scheme where only the faster *select*, *access*, and partial *rank* queries are used. This is described next.

4.2 Computing Scores Online

Firstly, we construct a supporting structure, SUP, in $O(k \lg^* n)$ time and occupying $o(n \lg D) + O(z \lg n)$ bits, capable of answering the following query in $O(\lg \lg^* n)$ time: for any given r, return $score(r, q')$ if $r \in F(q', k)$, otherwise return -1. Let $\Delta = \Theta(\lg^* n)$, then structure SUP is a forest of D/Δ balanced binary search trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{D/\Delta}$. Initially each $\mathcal{T}_i$ is empty, hence the initial space is $O(\lg n)$ bits per tree (for maintaining a pointer to the location where it is stored), adding up to $O((D/\Delta) \lg n) = o(n \lg D)$ bits, which we consider a part of index. Next we shall insert each document $d_r \in F(q', k)$, along with its associated score, into tree $\mathcal{T}_{\lceil r/\Delta \rceil}$ of SUP. The size of each search tree can grow up to Δ, hence the total insertion time is $O(k \lg \Delta)$. These insertions will increase the space of SUP by $O(k \lg n)$ bits, which can be justified as it is the size of the output. Now we can search for any d_r in $\mathcal{T}_{r/\Delta}$ and, if $d_r \in F(q', k)$, we will retrieve $score(r, q')$ in $O(\lg \Delta)$ time. Once we finish Step 3, these binary search trees can be set back to their initial empty state by visiting each document $d_r \in F(q', k)$ and deleting it from the corresponding tree in total $O(k \lg \Delta)$ time. This does not impact the total asymptotic query processing time.

An outline of Step 3 follows: We scan each leaf $\ell_i \in L(q \backslash q')$, and compute $score(\cdot, q)$ of the corresponding document $d_{DA[i]}$. Note that there can be many leaves in $L(q \backslash q')$ marked with the same document, but we compute $score(\cdot, q)$ of a document only once (i.e., when we encounter it for the first time). After this, we also scan the documents $d_r \in F(q', k)$ and compute $score(r, q)$ if we have not considered this document in the previous step. However, the scanning of leaves is performed in a carefully chosen order. Let $\ell_{sp'}$ and $\ell_{ep'}$ be the leftmost and rightmost leaves in the subtree of q', and $B[1..D]$ be a bit vector initialized to all 0's (its size is D bits and can be considered a part of index). A detailed description of Step 3 follows:

3.1 Start scanning the leaves ℓ_i for $i = sp, sp+1, \ldots, sp'-1$, in the *ascending* order of i, then for $i = ep, ep-1, \ldots, ep'+1$, in the *descending* order of i, and do the following: if $B[DA[i]] = 0$, then set it to 1, compute $score(DA[i], q)$, and store the result $(DA[i], score(DA[i], q))$ for Step 4. Note that each time we compute $score(DA[i], q)$, i is either the first or the last occurrence of $DA[i]$ in $DA[sp, ep]$. Assume it is the first (the other case is symmetric). We use a constant-time partial *rank* query, $x = rank_{DA}(DA[i], i)$. Then, by performing successive $select_{DA}(DA[i], j)$ queries for $j = x+1, x+2, \ldots, y$, where $select_{DA}(DA[i], y) > ep \geq select_{DA}(DA[i], y-1)$, we compute $score(DA[i], q) = y - x$. The number of *select* queries required is precisely $y - x = score(DA[i], q)$, which can be further reduced as follows:
   - If $d_{DA[i]} \in F(q', k)$, retrieve $score(DA[i], q')$ from SUP in time $O(\lg \lg^* n)$. As we know that $score(DA[i], q') \leq score(DA[i], q)$, we start *select* queries

from $j = x + \mathit{score}(DA[i], q')$, so the number of *select* queries used to find y is reduced to $\mathit{score}(DA[i], q) - \mathit{score}(DA[i], q') = \mathit{score}(DA[i], L(q \backslash q'))$, that is, the number of leaves in $L(q \backslash q')$ marked with $d_{DA[i]}$.

– If $d_{DA[i]} \notin F(q', k)$, compute $x' = \mathit{select}_{DA}(DA[i], x + \tau - 1)$, where we remind that $\tau = \min\{\mathit{score}(r, q'), r \in F(q', k)\}$. If $x' > ep$, we conclude that $\mathit{score}(DA[i], q) < \tau$, and hence $d_{DA[i]}$ can be discarded from being a candidate for the final output. On the other hand, if $x' \leq ep$, the *select* queries can be started from $j = x + \tau$, which reduces the number of *select* queries to $\mathit{score}(DA[i], q) - \tau \leq \mathit{score}(DA[i], L(q \backslash q'))$ (since $d_{DA[i]} \notin F(q', k)$, it holds $\mathit{score}(DA[i], q') \leq \tau$).

The query time for executing this step can be analyzed as follows: for each i, we perform a query on SUP. The computation of $\mathit{score}(DA[i], q)$ requires at most $\mathit{score}(DA[i], L(q \backslash q'))$ *select* queries. As we do this computation only once per distinct document, the total number of *select* queries is at most $\sum_r \mathit{score}(r, L(q \backslash q')) = |L(q \backslash q')|$. By choosing the cost $f(n, D) = \sqrt{\lg^* n}$ for *select* queries, the total time is $O(|L(q \backslash q')|(f(n, D) + \lg\lg^* n)) = O(k \lg^* n)$.

3.2 Now scan the documents $d_r \in F(q', k)$. If $B[r] = 0$, then there exists no leaf in $L(q \backslash q')$ marked with d_r. Thus $\mathit{score}(r, q) = \mathit{score}(r, q')$ and the pair $(r, \mathit{score}(r, q'))$ is stored for Step 4. If $B[r] = 1$ then d_r has already been dealt with in the previous pass. The time for accessing $\mathit{score}(r, q')$ using SUP is $O(\lg\lg^* n)$, hence this step takes $O(k \lg\lg^* n)$ time.

3.3 Reset B to its initial state (all bits set to 0) for supporting queries in future. By revisiting the leaves in $L(q \backslash q')$ and the list $F(q', k)$, we can exactly find out those locations in B where the corresponding bit is 1. The time for this step can be bounded by $O(|L(q \backslash q')| + k) = O(k\sqrt{\lg^* n})$.

Thus the time for Step 3 is $O(k \lg^* n)$, and the result follows.

5 Reducing the Time to $O(p + k \lg^* k)$

Note that, when p or k is at least $\lg\lg n$, it already holds $O(p + k \lg^* n) = O(p + k \lg^* k)$. Therefore, we now concentrate on the case when $\max(p, k) < \lg\lg n$. We use the following result [8].

Lemma 7. *Given a fixed κ, an array $A[1..n]$ of n indices can be indexed in $O(n \lg^2 \kappa)$ bits for answering the following query in $O(k)$ time, without accessing A and for any $1 \leq k \leq \kappa$: given i, j, and k, output the positions of the k highest elements in $A[i, j]$.*

Let S_δ be the set of nodes in GST with node depth equal to δ. We start with the description of an $O(n \lg^2 \kappa)$-bit structure for a fixed $\kappa = \lg\lg n$ and a fixed $\delta < \lg\lg n$, for answering top-k queries for any $1 \leq k \leq \kappa$ and those patterns with their locus node belonging to S_δ. First, we construct an array $A[1..n]$ (with all its elements initialized to zero) as follows: For $i = 1 \ldots n$, if the first occurrence of document $DA[i]$ in $DA[a, b]$ is at position i, where $[a, b]$ is the suffix range corresponding to a unique node $u \in S_\delta$, then set $A[i] = \mathit{score}(DA[i], u)$. We do not store this array explicitly, instead we maintain the structure of Lemma 7 over it, requiring $O(n \lg^2 \kappa)$ bits space. Now the list of documents $F(u, k)$ for any locus node $u \in S_\delta$ can be reported in $O(k)$ time as follows: First perform a top-k query on the structure of Lemma 7 with the suffix range $[sp, ep]$. The output will be a set of k locations $j_1, j_2, \ldots, j_k \in [sp, ep]$, and then the identifiers of the top-k documents are $DA[j_1], DA[j_2], \ldots, DA[j_k]$. By maintaining similar structures for all the $\delta \in [1, \lg\lg n)$, any such top-k query with $p < \lg\lg n$ can be answered in $O(p + k)$ time. The additional space required is $o(n(\lg\lg n)^3)$ bits, which can be bounded by $o(n \lg \sigma)$ bits if, say, $\lg \sigma \geq \sqrt{\lg n}$. Otherwise, we shall explicitly maintain the top-κ documents corresponding to all patterns of length at most $\lg\lg n$, in decreasing frequency order, using a table of $O(\sigma^{\lg\lg n} \lg\lg n \lg D) = o(n)$ bits. The query time in this case is just $O(k)$.

Thus, by combining the cases, we achieve $O(p + k \lg^* k)$ query time.

Theorem 2. *There exists a compact index of* $|\mathsf{CSA}| + n \lg D + o(n(\lg \sigma + \lg D))$ *bits and near-optimal* $O(p + k \lg^* k)$ *query time time, for the (unsorted) top-k frequent document retrieval problem.*

6 Conclusions

We have shown that it is possible to obtain almost optimal time for top-k document retrieval, $O(p+k \lg^* n)$, using compact space, $|\mathsf{CSA}|+n \lg D+o(n \lg D)$ bits. By adding $o(n \lg \sigma)$ bits, the time decreases to $O(p+k \lg^* k)$. This is an important step towards answering the question of which is the minimum space that is necessary to obtain the optimal time, $O(p+k)$. The other important open question is which is the minimum time that can be obtained by using the asymptotically optimal space, $|\mathsf{CSA}| + o(n)$ bits. Right now this time is $O(p + k \lg^2 k \lg^{1+\epsilon} n)$ [20], and it is not clear which is the lower bound.

References

1. Belazzougui, D., Navarro, G.: Alphabet-independent compressed text indexing. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 748–759. Springer, Heidelberg (2011)
2. Belazzougui, D., Navarro, G.: New lower and upper bounds for representing sequences. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 181–192. Springer, Heidelberg (2012)
3. Belazzougui, D., Navarro, G., Valenzuela, D.: Improved compressed indexes for full-text document retrieval. J. Discr. Alg. 18, 3–13 (2013)
4. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Comp. Sys. Sci. 7(4), 448–461 (1973)
5. Büttcher, S., Clarke, C., Cormack, G.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press (2010)
6. Culpepper, J.S., Navarro, G., Puglisi, S.J., Turpin, A.: Top-k ranked document search in general text databases. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 194–205. Springer, Heidelberg (2010)
7. Gagie, T., Kärkkäinen, J., Navarro, G., Puglisi, S.J.: Colored range queries and document retrieval. Theoretical Computer Science 483, 36–50 (2013)

8. Grossi, R., Iacono, J., Navarro, G., Raman, R., Rao, S.S.: Encodings for range selection and top-k queries. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 553–564. Springer, Heidelberg (2013)
9. Hon, W.-K., Patil, M., Shah, R., Wu, S.-B.: Efficient index for retrieving top-k most frequent documents. J. Discr. Alg. 8(4), 402–417 (2010)
10. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: Document listing for queries with excluded pattern. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 185–195. Springer, Heidelberg (2012)
11. Hon, W.-K., Shah, R., Thankachan, S., Vitter, J.: Faster compressed top-k document retrieval. In: Proc. 23rd DCC, pp. 341–350 (2013)
12. Hon, W.-K., Shah, R., Vitter, J.: Space-efficient framework for top-k string retrieval problems. In: Proc. 50th FOCS, pp. 713–722 (2009)
13. Hon, W.-K., Shah, R., Wu, S.-B.: Efficient index for retrieving top-k most frequent documents. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 182–193. Springer, Heidelberg (2009)
14. Konow, R., Navarro, G.: Faster compact top-k document retrieval. In: Proc. 23rd DCC, pp. 351–360 (2013)
15. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comp. 22(5), 935–948 (1993)
16. Muthukrishnan, S.: Efficient algorithms for document retrieval problems. In: Proc 13th SODA, pp. 657–666 (2002)
17. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comp. Surv. 39(1), art. 2 (2007)
18. Navarro, G., Nekrich, Y.: Top-k document retrieval in optimal time and linear space. In: Proc. 23rd SODA, pp. 1066–1078 (2012)
19. Navarro, G., Puglisi, S.J., Valenzuela, D.: Practical compressed document retrieval. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 193–205. Springer, Heidelberg (2011)
20. Navarro, G., Thankachan, S.V.: Faster top-k document retrieval in optimal space. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 255–262. Springer, Heidelberg (2013)
21. Navarro, G., Valenzuela, D.: Space-efficient top-k document retrieval. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 307–319. Springer, Heidelberg (2012)
22. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: Proc. 9th ALENEX (2007)
23. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans. Alg. 3(4), art. 43 (2007)
24. Sadakane, K.: Succinct data structures for flexible text retrieval systems. J. Discr. Alg. 5, 12–22 (2007)
25. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: Proc. 21st SODA, pp. 134–149 (2010)
26. Shah, R., Sheng, C., Thankachan, S.V., Vitter, J.S.: Top-k document retrieval in external memory. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 803–814. Springer, Heidelberg (2013)
27. Tsur, D.: Top-k document retrieval in optimal space. Inf. Proc. Lett. 113(12), 440–443 (2013)
28. Välimäki, N., Mäkinen, V.: Space-efficient algorithms for document retrieval. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 205–215. Springer, Heidelberg (2007)
29. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)

Faster, Space-Efficient Selection Algorithms in Read-Only Memory for Integers

Timothy M. Chan[1], J. Ian Munro[1], and Venkatesh Raman[2]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
{tmchan,imunro}@uwaterloo.ca

[2] The Institute of Mathematical sciences, Chennai 600 113, India
vraman@imsc.res.in

Abstract. Starting with Munro and Paterson (1980), the selection or median-finding problem has been extensively studied in the read-only memory model and in streaming models. Munro and Paterson's deterministic algorithm and its subsequent refinements require at least polylogarithmic or logarithmic space, whereas the algorithms by Munro and Raman (1996) and Raman and Ramnath (1999) can be made to use just $O(1)$ storage cells but take $O(n^{1+\varepsilon})$ time for an arbitrarily small constant $\varepsilon > 0$.

In this paper, we show that faster selection algorithms in read-only memory are possible if the input is a sequence of integers. For example, one algorithm uses $O(1)$ storage cells and takes $O(n \lg U)$ time where U is the universe size. Another algorithm uses $O(1)$ storage cells and takes $O(n \lg n \lg\lg U)$ time. We also describe an $O(n)$-time algorithm for finding an approximate median using $O(\lg^{\varepsilon} U)$ storage cells.

All our algorithms are simple and deterministic. Interestingly, one of our algorithms is inspired by 'centroids' of binary trees and finds an approximate median by repeatedly invoking a textbook algorithm for the 'majority' problem. This technique could be of independent interest.

1 Introduction

The topic of this paper is the classical *selection* problem, where we want to find the k-th smallest element of an input sequence of n elements for a given number k. (The *median* corresponds to the $k = \lceil n/2 \rceil$ case.) Specifically, we are interested in space-efficient algorithms in the *read-only memory* model, where the input is a read-only array and we want to minimize the amount of extra space needed in addition to the input array.

Selection in read-only memory has been studied since Munro and Paterson's pioneering work on streaming algorithms [13]. Their deterministic algorithm runs in $O(n \lg_s n + n \lg s)$ time using $O(s)$ storage cells, for any given $s = \Omega(\lg^2 n)$. Frederickson [9] refined the running time to $O(n \lg_s n + n \lg^* s)$. (This bound has been recently improved [8] slightly in the extreme end when s approaches $n/\lg n$.) All these algorithms work by finding an 'approximate median' in one pass, which is used to find the exact median or the k-th smallest element over several passes. The space of one-pass streaming algorithms for approximate medians has been improved to $O(\lg n)$ cells by Greenwald and Khanna [11]; this could potentially extend the range of allowed values for s to $\Omega(\lg n)$.

However, for small, sublogarithmic space, the best time bound increases dramatically. Munro and Raman [14] gave an $O(2^s sn^{1+1/s})$-time algorithm, and this was improved to $O(sn^{1+1/s} \lg n)$ by Raman and Ramnath [15], for any s from 1 to $\lg n$.

If randomization is allowed, much better results are possible. Chan [6] provided a matching upper and lower bound of $\Theta(n \lg \lg_s n)$ on the expected running time for all s from 1 to n. Our focus here, however, will be on deterministic algorithms only.

All the above results are in the comparison model. In this paper, we study the selection problem in the setting where the input elements are integers in $[U] := \{1, 2, \ldots, U\}$. Our model of computation is the word RAM model where standard (arithmetic, shift and bitwise logical) word operations can be performed in constant time. We assume that each word is w bits long, and w is at least $\lg n$ and $\lg U$, so that a pointer or input integer can fit in a word. We assume that each memory cell can store a word.

The case of integer input is standard when studying general lower bounds, for example, for sorting in the read-only memory model [1,3], and randomized selection in the streaming model [5]; these lower bounds often match or nearly match their corresponding upper bounds in the comparison model. In contrast, we show that the integer assumption makes a big difference for the deterministic selection problem in read-only memory with small space. Our results are the following:

1. a selection algorithm using $O(n \lg_s U)$ time and $O(s)$ words of space for any s from 1 to n;
2. a selection algorithm using $O(n \lg n \lg_s \lg U)$ time and $O(s)$ words of space for any s from 1 to $\lg U$.

The first algorithm, presented in Section 2, is very simple and works well when the universe size U is polynomially bounded. For example, for $s = 1$ and $U = n^{O(1)}$, the running time is $O(n \lg n)$, which is significantly faster than those of Munro and Raman or Raman and Ramnath for constant s. For $s = \lg^{\varepsilon} n$ and $U = n^{O(1)}$, the running time is $O(n \lg n / \lg \lg n)$, where ε denotes an arbitrarily small positive constant. For $s = n^{\varepsilon}$ and $U = n^{O(1)}$, we get linear running time, eliminating the iterated logarithmic factor from Frederickson's result.

The second algorithm, presented in Section 3, is less sensitive to U and beats previous algorithms for a wider range of universe sizes. On our way to obtaining this second result, we also give

3. an approximate median algorithm using $O(n \lg_s \lg U)$ time and $O(s)$ words of space for any s from 1 to $\lg U$.

For $s = 1$, the time bound is $O(n \lg \lg U)$. For $s = \lg^{\varepsilon} U$, it is linear. The algorithm makes a clever use of a well-known algorithm, first published in [4] (discovered in 1980) to find the *majority* of a sequence of bits. To speed up this algorithm, we give a procedure to find the majority at several prefix lengths of the given sequence simultaneously in linear time. The resulting linear-time algorithm for approximate median is relatively simple and self-contained, and immediately implies a new deterministic linear-time algorithm for selection in the traditional, non-space-efficient setting, when the input elements are integers. This may be of independent interest, as the standard deterministic linear-time algorithm for selection [2] requires a doubly recursive structure, which the new algorithm manages to avoid.

When the two algorithms are combined, it is possible to obtain, for example, a selection algorithm that uses $O(1)$ words of space and whose running time is guaranteed to be at most $O(n \lg^{1+\varepsilon} n)$ regardless of the universe size U. This consequence is noted in Section 4.

We should mention that at least one prior work on approximate medians has also addressed upper bounds for the integer case: Shrivastava et al. [16] gave one-pass streaming algorithms for maintaining approximate quantiles using $O(\lg U)$ words of space. There are some similarities, but our approach is simpler, besides being more space-efficient (they maintained information about the trie induced by the binary representations of the numbers, whereas we maintain just one path in the trie, since we do not need all approximate quantiles). Like many other works in streaming algorithms, they were less interested in analyzing the total running time. It is conceivable that some of the previous streaming algorithms on approximate medians could be sped up in the integer case by using advanced data structures such as the fusion tree [10]. Our algorithms however do not require such complicated data structures.

2 An $O(n \lg_s U)$-Time Algorithm

We assume that the input integers are in the range $[U]$, and the aim is to find the k-th smallest element in the sequence of n integers. We give a very simple selection algorithm in read-only memory that takes $O(n \lg U)$ time using $O(1)$ space.

The algorithm goes through $\lg U$ stages where in stage i, we determine the i-th bit of the solution. At the first stage, we simply count the number of input integers with leading bit 0 and leading bit 1 and based on k, we determine the first bit of the answer, and update k to be the new rank of the element to be found among those input integers with the leading bit same as that of the answer.

We repeat the same procedure in the i-th stage, for $i > 1$ (having computed the $i-1$ bits of the answer) by counting how many integers, among those matching the first $i-1$ bits of the answer, have the i-th bit 0 and how many have the i-th bit 1, updating k and recursing appropriately. Thus we immediately obtain the following result:

Theorem 1. *Given a sequence of n integers in the range $[U]$, we can find the k-th smallest element of the sequence using $O(\lg U)$ passes, in $O(n \lg U)$ time using $O(1)$ words of space, in read-only memory.*

If we have $s = 2^b$ extra cells available, then we can read b bits of the input in each pass counting the number of inputs with each b-bit value using the 2^b counters, and finding b bits of the answer at each stage. The number of stages is thus reduced to $O((\lg U)/b)$. To ensure $O(n)$ run time for each pass, we limit the number of counters s to n. This results in the following theorem.

Theorem 2. *Given a sequence of n integers in the range $[U]$, we can find the k-th smallest element of the sequence using $O(\lg_s U)$ passes in $O(n \lg_s U)$ time using $O(s)$ words of space, in read-only memory for any $s \leq n$.*

3 An $O(n \lg n \lg_s \lg U)$-Time Algorithm

Our next approach solves the selection problem by designing better algorithms for finding an approximate median—specifically, an element whose rank is between $n/4$ and $3n/4$. The connection between exact selection and approximate median is well known. Given an algorithm for the latter problem, we can apply it to find an approximate median m for the subsequence of all input elements inside a current interval $[\ell, r]$ that contains the answer; we can then compute the rank of m in one additional pass, and then shrink $[\ell, r]$ to either $[\ell, m]$ or $[m, r]$, depending on whether the rank of m is greater or less than k. After about $\lg_{4/3} n$ iterations, the interval will be shrunk to a single point. If the given approximate median algorithm makes P passes over the input, has running time proportional to the cost of the P linear scans, and uses $O(s)$ space, then the overall algorithm has running time $O(nP \lg n)$ and uses $O(s)$ space.

We first present a simple algorithm to find an approximate median using $O(\lg \lg U)$ passes and $O(1)$ space. Our idea is inspired by the standard construction of a 'centroid' of a binary tree (namely, the trie formed by the binary representations of the input integers).

Theorem 3. *An approximate median from a range of $[U]$ can be found in $O(\lg \lg U)$ passes and $O(n \lg \lg U)$ time in read-only memory using $O(1)$ words of space. Hence the k-th smallest element can be found in $O(n \lg n \lg \lg U)$ time in read-only memory using $O(1)$ words of space.*

Proof. The idea is to find the longest prefix p such that the number of integers in the input sequence whose binary representations starts with p is more than $n/2$. Let m_0 and m_1 be the smallest input integer with prefix $p0$ and prefix $p1$ respectively, and let m_2 be the largest input integer with prefix $p1$. Let r_0, r_1, r_2 be the ranks of m_0, m_1, m_2 respectively. As $r_1 - r_0 \leq n/2$, $r_2 - r_1 \leq n/2$, and $r_2 - r_0 \geq n/2$ (due to the choice of p), it is easy to see that at least one of r_0, r_1, r_2 must lie in $[n/4, 3n/4]$. So, we can report one of m_0, m_1, m_2 as an approximate median.

Now, to find this longest prefix p, we use a standard binary-search over the lengths. We maintain an interval containing the desired prefix length (which is the entire length of $\lg U$ to start with). We take the midpoint of the interval, and if we find that this length has no element that occurs more than $n/2$ times (i.e., has no *majority* element), then we replace the interval with its lower half, else we replace the interval with its upper half. The number of iterations is $O(\lg \lg U)$.

At each prefix length, we need to decide whether a majority element exists in of a list of n elements. There is a well-known 'textbook' algorithm that solves the majority problem in $O(n)$ time, in two passes. We include a brief description, as it will be useful later. In the first pass, the pseudocode below finds a possible candidate p for the majority of $A[1], \ldots, A[n]$:

```
1. initialize $c = 0$
2. for $t = 1, \ldots, n$:
3.     if $c = 0$ then set $p = A[t]$
4.     if $A[t] = p$ then increment $c$ else decrement $c$
```

The claim is that after each iteration t,

($*$) if the majority of $A[1], \ldots, A[t]$ exists, then it must be p.

This can be seen from the following invariants: Let t' be the last value such that $c = 0$ at the end of iteration t'. Then

(a) $A[1], \ldots, A[t']$ do not have a majority; and
(b) $c = $ [# of times p occurs in $A[t'+1], \ldots, A[t]$] $-$
[# of times p does not occur in $A[t'+1], \ldots, A[t]$] ≥ 0.

By (b), the majority of $A[t'+1], \ldots, A[t]$ is p if $c > 0$, and does not exist if $c = 0$. Together with (a), this implies that ($*$), because of the following property: the majority of the concatenation of two lists, if it exists, must be the majority of one of the two lists.

After computing the candidate majority p, we can in a second pass compute its frequency and then check whether it is more than $n/2$. Once we find the longest prefix p, we find m_0, m_1 and m_2 and their ranks r_0, r_1 and r_2 respectively in two additional passes in $O(n)$ time. □

We show how to speed up the preceding algorithm when we allow a little more space:

Theorem 4. *An approximate median can be found in $O(n \lg_s \lg U)$ time using $O(s)$ words of space in read-only memory for any $s \leq \lg U$. Hence the k-th smallest element can be found in $O(n \lg n \lg_s \lg U)$ time using $O(s)$ words of space in read-only memory.*

Proof. We speed up the preceding binary search with an 's-ary search', which reduces the number of iterations to $O(\lg_s \lg U)$. This requires extending the majority algorithm to compute majority candidates for s length values simultaneously, in two passes. More precisely, let $A[1], \ldots, A[n]$ be the input array, let $\ell_1, \ldots, \ell_s$ be s given lengths in increasing order, and let $\pi_i(p)$ denote the length-ℓ_i prefix of the binary representation of p. We want to compute a candidate for the majority of $\pi_i(A[1]), \ldots, \pi_i(A[n])$ for every $i = 1, \ldots, s$.

The pseudocode for the modified majority algorithm is given below. The key is to observe that we can make the s majority candidates to be prefixes of one common string p. This requires one subtle twist in the algorithm, where a counter may in one particular case stay at zero without being incremented or decremented:

```
0. initialize $c_1, \ldots, c_s$ to 0, $p$ to 0
1. for $t = 1, \ldots, n$:
2.     find the smallest $j$ with $\pi_j(A[t]) \neq \pi_j(p)$
3.     if $c_j = 0$ (or $j$ does not exist) then
4.         set $p = A[t]$ and increment $c_1, \ldots, c_s$
5.     else increment $c_1, \ldots, c_{j-1}$
6.         decrement the *nonzero* entries of $c_j, \ldots, c_s$
```

Correctness. The claim is that after each iteration t, for each i,

($*$) if the majority of $\pi_i(A[1]), \ldots, \pi_i(A[t])$ exists, then it is $\pi_i(p)$, and $c_i \neq 0$.

Let t_i be the last value such that $c_i = 0$ at the end of iteration t_i. Then the following three invariants are true:

(a) $\pi_i(A[1]), \ldots, \pi_i(A[t_i])$ do not have a majority;
(b) $c_i =$ [# of times $\pi_i(p)$ occurs in $\pi_i(A[t_i+1]), \ldots, \pi_i(A[t])$] $-$
[# of times $\pi_i(p)$ does not occur in $\pi_i(A[t_i+1]), \ldots, \pi_i(A[t])$] ≥ 0;
(c) $c_1 \geq c_2 \geq \cdots \geq c_s$.

First we argue that ($*$) follows from the three invariants. By (b), the majority of $\pi_i(A[t_i+1]), \ldots, \pi_i(A[t])$ is $\pi_i(p)$ if $c_i > 0$, and does not exist if $c_i = 0$. Together with (a), this implies ($*$).

It is straightforward to verify that the invariants are preserved in the case when lines 3–4 are executed (here, $c_j = \cdots = c_b = 0$ by (c)). So consider the $c_j \neq 0$ case instead when lines 5–6 are executed. It is straightforward to see that the invariants are preserved for any index i where c_i is incremented or decremented. The remaining subcase is when $i > j$ and c_i is zero, and is not decremented, but stays at zero. Then a majority of $\pi_i(A[1]), \ldots, \pi_i(A[t-1])$ does not exist. To maintain (a), we need to confirm that a majority of $\pi_i(A[1]), \ldots, \pi_i(A[t])$ does not exist. Assume otherwise. Then this majority must be $\pi_i(A[t])$. This would imply that the majority of $\pi_j(A[1]), \ldots, \pi_j(A[t])$ is $\pi_j(A[t])$, but this majority can only be $\pi_j(p)$: a contradiction with $\pi_j(A[t]) \neq \pi_j(p)$.

Implementation. Line 2 can be performed in $O(1)$ time by taking the exclusive-or of $A[t]$ and p and identifying the most significant 1-bit (a commonly encountered operation, which is known to be reducible to $O(1)$ standard word operations [10]). Lines 4, 5, and 6 require increment/decrement operations on length-s vectors, which can be done using a known data structure for dynamic counting by Dietz [7]. Because s is small, however, the data structure can be greatly simplified and we can give a short description.

The idea is to express $(c_s, \ldots, c_1)$ as a sum $(c'_s, \ldots, c'_1) + (\delta_s, \ldots, \delta_1)$, where the first vector does not change often and the second vector is kept small. Specifically, after a round of $s-1$ iterations, we reset $c'_i = \max\{c_i - s, 0\}$ and $\delta_i = c_i - c'_i \geq 0$;

the amortized cost of the reset is $O(1)$. During a round, updates are applied to the δ_i's. Then $\delta_i < 2s$ at all times, so $(\delta_s, \ldots, \delta_1)$ can be packed in one word provided that $s \lg s = o(\lg U)$. If $s \lg s = \Omega(\lg U)$, we can change s to $\sqrt{\lg U}$, for example, and the result is the same because $O(n \lg_s \lg U)$ is the same as $O(n)$. Testing whether $c_i = 0$ is equivalent to testing whether $\delta_i = 0$. Line 4 corresponds to adding a constant $(1, \ldots, 1)$ to $(\delta_s, \ldots, \delta_1)$ and lines 5–6 correspond to adding a vector of the form $(0, \ldots, 0, 1, \ldots, 1)$ and subtracting a vector of the form $(0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0)$. These reduce to $O(1)$ standard arithmetic operations on words. Before we decrement, we need to identify the smallest i with $\delta_i = 0$; this reduces to finding the most significant 1-bit in $(\delta_s, \ldots, \delta_1)$.

Thus, the algorithm can be implemented to run in $O(n)$ time. It uses $O(s)$ words of space for storing p and $(c_s, \ldots, c_1)$.

After computing the candidate majorities $\pi_i(p)$, we can in a second pass compute their frequencies f_i: initialize $f_1, \ldots, f_s$ to 0, and for each $t = 1, \ldots, n$, find the smallest j with $\pi_j(A[t]) \neq \pi_j(p)$ and increment $f_1, \ldots, f_{j-1}$. By the same approach, the second pass takes $O(n)$ time as well. We can then check which of the frequencies are more than $n/2$. □

4 Final Remarks

We have shown that the selection problem for integers in read-only memory can be solved deterministically in $O(\min\{n\lceil \lg_s U\rceil,\ n \lg n \lceil \lg_s \lg U\rceil\})$ time with $O(s)$ storage cells. Actually in all our algorithms, all but a constant number of cells in the extra memory store $O(\lg n)$-bit pointers or counters rather than $O(\lg U)$-bit integers. Thus, the space used in bits is $O(\lg U + s \lg n)$.

For example, we can set $s = \lg U / \lg n$ to ensure $O(\lg U)$ bits of space, and a time bound of $O(\min\left\{n \frac{\lg U}{\lg(\lg U/\lg n)},\ n \lg n \frac{\lg\lg U}{\lg(\lg U/\lg n)}\right\})$. Notice that the second term is $O(n \lg n)$ when $\lg U \geq \lg^{1+\varepsilon} n$. On the other hand, when $\lg U \leq \lg^{1+\varepsilon} n$, the first term is at most $O(n \lg^{1+\varepsilon} n)$. So, a combination of our two approaches yields an algorithm that uses $O(1)$ words of space and always runs in at most $O(n \lg^{1+\varepsilon} n)$ time—a bound independent of U.

A remaining question is whether there is a deterministic selection algorithm for integers in read-only memory that uses $O(1)$ words of space and runs in $O(n \lg n)$ time (or better) for all U. Another question is to what extent can our approximate median algorithm be improved; for example, is there a deterministic linear-time algorithm using $O(1)$ words of space in read-only memory? Also, in the comparison model, the best deterministic algorithm with $O(1)$ space currently requires $O(n^{1+\varepsilon})$ time in read-only memory [14,15]. Can one prove a matching lower bound? This question appears difficult.

Finally, as was mentioned in the introduction, if randomization is allowed, $\Theta(n \lg \lg_s n)$ expected time algorithm is possible using $O(s)$ storage cells for general input. Can one extend the $\Omega(n \lg \lg_s n)$ lower bound proof [6] to a non-comparison model in the case when the input is a sequence of integers?

References

1. Beame, P.: A general sequential time-space tradeoff for finding unique elements. SIAM Journal on Computing 20(2), 270–277 (1991)
2. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. Journal of Computer and System Sciences 7, 448–461 (1973)
3. Borodin, A., Cook, S.A.: A time-space tradeoff for sorting on a general sequential model of computation. SIAM Journal on Computing 11, 287–297 (1982)
4. Boyer, R.S., Moore, J.S.: MJRTY - A fast majority vote algorithm. In: Boyer, R.S. (ed.) Automated Reasoning: Essays in Honor of Woody Bledsoe. Automated Reasoning Series, pp. 105–117. Kluwer (1991)
5. Chakrabarti, A., Jayram, T.S., Pătraşcu, M.: Tight lower bound for selection in randomly ordered streams. In: Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 720–729 (2008)
6. Chan, T.M.: Comparison-based time-space lower bounds for selection. ACM Transactions on Algorithms 6(2), 26 (2010)
7. Dietz, P.F.: Optimal algorithms for list indexing and subset rank. In: Dehne, F., Santoro, N., Sack, J.-R. (eds.) WADS 1989. LNCS, vol. 382, pp. 39–46. Springer, Heidelberg (1989)
8. Elmasry, A., Juhl, D.D., Katajainen, J., Satti, S.R.: Selection from read-only memory with limited work space. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 147–157. Springer, Heidelberg (2013)
9. Frederickson, G.: Upper bounds for time-space trade-offs in sorting and selection. Journal of Computer and System Sciences 34(1), 19–26 (1987)
10. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. Journal of Computer and System Sciences 47, 424–436 (1993)
11. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: Proceedings of ACM SIGMOD, pp. 58–66 (2001)
12. Manku, G.S., Rajagopalan, S., Lindsay, B.G.: Approximate medians and other quantiles in one pass with limited memory. Proceedings of ACM SIGMOD 27(2), 426–435 (1998)
13. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theoretical Computer Science 12, 315–323 (1980)
14. Munro, J.I., Raman, V.: Selection from read-only memory and sorting with optimum data movement. Theoretical Computer Science 165(2), 311–323 (1996)
15. Raman, V., Ramnath, S.: Improved upper bounds for time-space tradeoffs for selection. Nordic Journal of Computing 6(2), 162–180 (1999)
16. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: new aggregation techniques for sensor networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 239–249 (2004)

Trajectory-Based Dynamic Map Labeling

Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg

Institute of Theoretical Informatics,
Karlsruhe Institute of Technology, Germany

Abstract. In this paper we introduce *trajectory-based labeling*, a new variant of dynamic map labeling where a movement trajectory for the map viewport is given. We define a general labeling model and study the active range maximization problem in this model. The problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard. In the restricted, yet practically relevant case that no more than k labels can be active at any time, we give polynomial-time algorithms. For the general case we present a practical ILP formulation with an experimental evaluation as well as approximation algorithms.

1 Introduction

In contrast to traditional static maps, dynamic digital maps support continuous movement of the map viewport based on panning, rotation, or zooming. Creating smooth visualizations under such map dynamics induces challenging geometric problems, e.g., continuous generalization [12] or dynamic map labeling [2]. In this paper, we focus on map labeling and take a trajectory-based view on it. In many applications, e.g., car navigation, a movement trajectory is known in advance and it becomes interesting to optimize the visualization of the map locally along this trajectory.

Selecting and placing a maximum number of non-overlapping labels for various map features is an important cartographic problem. Labels are usually modeled as rectangles and a typical objective in a static map is to find a maximum (possibly weighted) independent set of labels. This is known to be $\mathcal{NP}$-complete [6]. There are several approximation algorithms and PTAS's in different labeling models [1, 5], as well as practically useful heuristics [13, 14].

With the increasing popularity of interactive dynamic maps, e.g., as digital globes or on mobile devices, the static labeling problem has been translated into a dynamic setting. Due to the temporal dimension of the animations occurring during map movement, it is necessary to define a notion of *temporal consistency* or *coherence* for map labeling as to avoid distracting effects such as jumping or flickering labels [2]. Previously, consistent labeling has been studied from a global perspective under continuous zooming [3] and continuous rotation [8]. In practice, however, an individual map user with a mobile device, e.g., a tourist or a car driver, is typically interested only in a specific part of a map and it is thus often more important to optimize the labeling locally for a certain trajectory of the map viewport than globally for the whole map.

We introduce a versatile trajectory-based model for dynamic map labeling, and define three label activity models that guarantee consistency. We apply this model to point feature labeling for a viewport that moves and rotates along a differentiable trajectory in a fixed-scale base map in a forward-facing way. Although we present our approach in

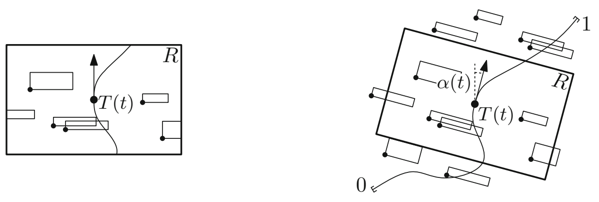


Fig. 1. Illustration of the viewport moving along a trajectory. Left the user's view and right a general view of the map and the viewport.

a very specific problem setting, our model is very general. Our approach can be applied for every dynamic labeling problem that can be expressed as a set of label availability intervals over time and a set of conflict intervals over time for pairs of labels. The exact algorithms hold for the general model, the approximation algorithm itself is also applicable, but the analysis of the approximation ratio requires problem-specific geometric arguments, which must be adjusted to the specific setting.

Contribution. For our specific problem, we show that maximizing the number of visible labels integrated over time in our model is $\mathcal{NP}$-complete; in fact it is even $\mathcal{W}[1]$-hard and thus it is unlikely that a fixed-parameter tractable algorithm exists. We present an integer linear programming (ILP) formulation for the general unrestricted case, which is supported by a short experimental evaluation. For the special case of unit-square labels we give an efficient approximation algorithm with different approximation ratios depending on the actual label activity model. Moreover, we present polynomial-time algorithms for the restricted case that no more than k labels are active at any time for some constant k. We note that limiting the number of simultaneously active labels is of practical interest as to avoid overly dense labelings, in particular for dynamic maps on small-screen devices such as in car navigation systems. Due to space constraints we omitted some proofs, which can be found in the full version [7] of the paper.

2 Trajectory-Based Labeling Model

Let M be a labeled north-facing, fixed-scale map, i.e., a set of points $P = \{p_1, \dots, p_N\}$ in the plane together with a corresponding set $L = \{\ell_1, \dots, \ell_N\}$ of labels. Each label ℓ_i is represented by an axis-aligned rectangle of individual width and height. We call the point p_i the *anchor* of the label ℓ_i. Here we assume that each label has an arbitrary but fixed position relative to its anchor, e.g., with its lower left corner coinciding with the anchor. The *viewport* R is an arbitrarily oriented rectangle of fixed size that defines the currently visible part of M on the map screen. The viewport follows a trajectory that is given by a continuous differentiable function $T\colon [0,1] \to \mathbb{R}^2$. For an example see Fig. 1. More precisely, we describe the viewport by a function $V\colon [0,1] \to \mathbb{R}^2 \times [0, 2\pi]$. The interpretation of $V(t) = (c, \alpha)$ is that at time t the center of the rectangle R is located at c and R is rotated clockwise by the angle α relatively to a north base line of the map. Since R moves along T we define $V(t) = (T(t), \alpha(t))$, where $\alpha(t)$ denotes the direction of T at time t. For simplicity, we sometimes refer to R at time t as $V(t)$. To ensure good readability, we require that the labels are always aligned with the viewport axes as the viewport changes its orientation, i.e., they rotate around their anchors by the same angle $\alpha(t)$, see Fig. 1. We denote the rotated label rectangle of ℓ at time t by $\ell(t)$.

We say that a label ℓ is *present* at time t, if $V(t) \cap \ell(t) \neq \emptyset$. As we consider the rectangles $\ell(t)$ and $V(t)$ to be closed, we can describe the points in time for which ℓ is present by closed intervals. We define for each label ℓ the set Ψ_ℓ that describes all disjoint subintervals of $[0,1]$ for which ℓ is present, thus $\Psi_\ell = \{[a,b] \mid [a,b] \subseteq [0,1]$ is maximal so that ℓ is present at all $t \in [a,b]\}$. Further, we define the disjoint union $\Psi = \{([a,b],\ell) \mid [a,b] \in \Psi_\ell$ and $\ell \in L\}$ of all Ψ_ℓ. We abbreviate $([a,b],\ell) \in \Psi$ by $[a,b]_\ell$ and call $[a,b]_\ell \in \Psi$ a *presence interval* of ℓ. In the remainder of this paper we denote the number of presence intervals by n.

Two labels ℓ and ℓ' are in *conflict* with each other at time t if $\ell(t) \cap \ell'(t) \neq \emptyset$. If $\ell(t) \cap \ell'(t) \cap V(t) \neq \emptyset$ we say that the conflict is *present* at time t. As in [8] we can describe the occurrences of conflicts between two labels $\ell, \ell' \in L$ by a set of closed intervals: $C_{\ell,\ell'} = \{[a,b] \subseteq [0,1] \mid [a,b]$ is maximal and ℓ and ℓ' are in conflict at all $t \in [a,b]\}$. We define the disjoint union $C = \{([a,b],\ell,\ell') \mid [a,b] \in C_{\ell,\ell'}$ and $\ell,\ell' \in L\}$ of all $C_{\ell,\ell'}$. We abbreviate $([a,b],\ell,\ell') \in C$ as $[a,b]_{\ell,\ell'}$ and call it a *conflict interval* of ℓ and ℓ'. Two presence intervals $[a,b]_\ell$ and $[c,d]_{\ell'}$ *are in conflict* if there is a conflict $[f,g]_{\ell,\ell'} \in C$ s.t. the intersection of the intervals $[f,g]_{\ell,\ell'} \cap [a,b]_\ell \cap [c,d]_{\ell'} \neq \emptyset$.

The tuple (P, L, Ψ, C) is called an *instance* of trajectory-based labeling. Note that the essential information of T is implicitly given by Ψ and C and that for each label $\ell \in L$ there can be several presence intervals. In this paper we assume that Ψ and C is given as input. In practice, however, we usually first need to compute Ψ and C given a continuous and differentiable trajectory T. An interesting special case is that T is a continuous, differentiable chain of m circular arcs (possibly of infinite radius), e.g., obtained by approximating a polygonal route in a road network. Niedermann [11] showed that in this case the set Ψ can be computed in $O(m \cdot N)$ time and the set C in $O(m \cdot N^2)$ time. His main observation was that for each arc of T the viewport can in fact be treated as a huge label and that "conflicts" with the viewport correspond to presence intervals. We refer to [11, Chapter 15] for details.

Next we define the *activity* of labels, i.e., when to actually display which of the present labels on screen. We restrict ourselves to closed and disjoint intervals describing the activity of a label ℓ and define the set $\Phi_\ell = \{[a,b] \subseteq [0,1] \mid [a,b]$ is maximal such that ℓ is active at all $t \in [a,b]\}$, as well as the disjoint union $\Phi = \{([a,b],\ell) \mid [a,b] \in \Phi_\ell$ and $\ell \in L\}$ of all Φ_ℓ. We abbreviate $([a,b],\ell) \in \Phi$ with $[a,b]_\ell$ and call $[a,b]_\ell \in \Phi$ an *active interval* of ℓ.

It remains to define an *activity model* restricting Φ in order to obtain a reasonable labeling. Here we propose three activity models AM1, AM2, AM3 with increasing flexibility. All three activity models exclude overlaps of displayed labels and guarantee consistency criteria introduced by Been et al. [2], i.e., labels must not flicker or jump. To that end they share the following properties **(A)** a label ℓ can only be active at time t if it is present at time t, **(B)** to avoid flickering and jumping each presence interval of ℓ contains at most one active interval of ℓ, and **(C)** if two labels are in conflict at a time t, then at most one of them may be active at t to avoid overlapping labels.

What distinguishes the three models are the possible points in time when labels can become active or inactive. The first and most restrictive activity model AM1 demands that each activity interval $[a,b]_\ell$ of a label ℓ must coincide with a presence interval of ℓ. The second activity model AM2 allows an active interval of a label ℓ to end earlier than the corresponding presence interval if there is a *witness label* ℓ' for that, i.e., an active interval for ℓ may end at time c if there is a starting conflict interval $[c,d]_{\ell,\ell'}$ and the conflicting label ℓ' is active at c. However, AM2 still requires every active interval to begin with the corresponding presence interval. The third activity model AM3 extends AM2 by also relaxing the restriction regarding the start of active intervals. An active interval for a label ℓ may start at time c if a present conflict $[a,c]_{\ell,\ell'}$ involving ℓ and an active witness label ℓ' ends at time c. In this model active intervals may begin later and end earlier than their corresponding presence intervals if there is a visible reason for the map user to do so, namely the start or end of a conflict with an active witness label.

A common objective in both static and dynamic map labeling is to maximize the number of labeled points. Often, however, certain labels are more important than others. To account for this, each label ℓ can be assigned a weight W_ℓ that corresponds to its significance. Then we define the weight of an interval $[a,b]_\ell \in \Phi$ as $w([a,b]_\ell) = (b-a)\cdot W_\ell$. Given an instance (P,L,Ψ,C), then with respect to one of the three activity models we want to find an activity Φ that maximizes $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell)$; we call this optimization problem GENERALMAXTOTAL. If we require that at any time t at most k labels are active for some k, we call the problem k-RESTRICTEDMAXTOTAL. In particular the latter problem is interesting for small-screen devices, e.g., car navigation systems, that should not overwhelm the user with additional information.

3 Solving GENERALMAXTOTAL

We first prove that GENERALMAXTOTAL is $\mathcal{NP}$-complete. The membership of GENERALMAXTOTAL in $\mathcal{NP}$ follows from the fact that the start and the end of an active interval must coincide with the start or end of a presence interval or a conflict interval. Thus, there is a finite number of candidates for the endpoints of the active intervals so that a solution $\mathcal{L}$ can be guessed. Verifying that $\mathcal{L}$ is valid in one of the three models and that its value exceeds a given threshold can obviously be checked in polynomial time.

For the $\mathcal{NP}$-hardness we apply a straight-forward reduction from the $\mathcal{NP}$-complete maximum independent set of rectangles problem [6]. We simply interpret the set of rectangles as a set of labels with unit weight, choose a short vertical trajectory T and a viewport R that contains all labels at any point of T. Since the conflicts do no change over time, the reduction can be used for all three activity models. By means of the same reduction and Marx' result [10] that finding an independent set for a given set of axis-parallel unit squares is $\mathcal{W}[1]$-hard we derive the next theorem.

Theorem 1. GENERALMAXTOTAL *is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard for all activity models AM1–AM3.*

As a consequence, GENERALMAXTOTAL is not fixed-parameter tractable unless $\mathcal{W}[1] = \mathcal{FPT}$. Note that this also means that for k-RestrictedMaxTotal we cannot expect to find an algorithm that runs in $O(p(n)\cdot C(k))$ time, where $p(n)$ is a polynomial that depends only on the number n of presence intervals and the computable function $C(k)$ depends only on the parameter k.

3.1 Integer Linear Programming for GENERALMAXTOTAL

Since we are still interested in finding an optimal solution for GENERALMAXTOTAL we have developed integer linear programming (ILP) formulations for all three activity models. We present the formulation for the most involved model AM3 and then argue how to adapt it to the simpler models AM1 and AM2.

We define E to be the totally ordered set of the endpoints of all presence and all conflict intervals and include 0 and 1; see Fig. 2. We call each interval $[c, d]$ between two consecutive elements c and d in E an *atomic segment* and denote the i-th atomic segment of E by $E(i)$. Further, let $X(\ell, i)$ be the set of labels that are in conflict with ℓ during $E(i-1)$, but not during $E(i)$, i.e., the conflicts end with $E(i-1)$. Analogously, let $Y(\ell, i)$ be the set of labels that are in conflict with ℓ during $E(i+1)$, but not during $E(i)$, i.e., the conflicts begin with $E(i+1)$. For each label ℓ we introduce three binary variables $b_i, x_i, e_i \in \{0, 1\}$ and the following constraints.

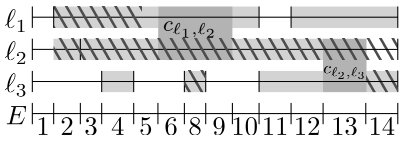


Fig. 2. Depiction of presence intervals (light gray), active intervals (hatched), and conflicts (dark gray)

$$b_i^\ell = x_i^\ell = e_i^\ell = 0 \qquad \forall 1 \le i \le |E| \text{ s.t. } \forall [c,d] \in \Psi_\ell : E(i) \cap [c,d] = \emptyset \tag{1}$$

$$\sum_{j \in J} b_j^\ell \le 1 \text{ and } \sum_{j \in J} e_j^\ell \le 1 \qquad \forall [c,d] \in \Psi_\ell \text{ where } J = \{j \mid E(j) \subseteq [c,d]\} \tag{2}$$

$$x_i^\ell + x_i^{\ell'} \le 1 \qquad \forall 1 \le i \le |E| \; \forall [c,d]_{\ell,\ell'} \in C : \; E(i) \subseteq [c,d] \tag{3}$$

$$x_{i-1}^\ell + b_i^\ell = x_i^\ell + e_{i-1}^\ell \qquad \forall 1 \le i \le |E| \text{ (set } x_0 = e_0 = 0) \tag{4}$$

$$b_j^\ell \le \sum_{\ell' \in X(\ell,j)} x_{j-1}^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } c \notin E(j) \tag{5}$$

$$e_j^\ell \le \sum_{\ell' \in Y(\ell,j)} x_{j+1}^{\ell'} \qquad \forall [c,d]_\ell \in \Psi \; \forall E(j) \subset [c,d]_\ell \text{ with } d \notin E(j) \tag{6}$$

Subject to these constraints we maximize $\sum_{\ell \in L} \sum_{i=1}^{|E|-1} x_i^\ell \cdot w(E(i))$. The intended meaning of the variables is that $x_i^\ell = 1$ if ℓ is active during $E(i)$ and otherwise $x_i^\ell = 0$. Variable $b_i^\ell = 1$ if and only if $E(i)$ is the first atomic segment of an active interval of ℓ, and analogously $e_i^\ell = 1$ if and only if $E(i)$ is the last atomic segment of an active interval of ℓ. Recall the properties of the activity models as defined in Section 2. Constraints (1)–(3) immediately ensure properties (A)–(C), respectively. Constraint (4) means that if ℓ is active during $E(i-1)$ ($x_{i-1}^\ell = 1$), then it must either stay active during $E(i)$ ($x_i^\ell = 1$) or the active interval ends with $E(i-1)$ ($e_{i-1}^\ell = 1$), and if ℓ is active during $E(i)$ ($x_i^\ell = 1$) then it must be active during $E(i-1)$ ($x_{i-1}^\ell = 1$) or the active interval begins with $E(i)$ ($b_i^\ell = 1$). Constraint (5) enforces that for ℓ to become active with $E(j)$ at least one witness label of $X(\ell, j)$ is active during $E(j-1)$. Analogously, constraint (6) enforces that for ℓ to become inactive with $E(j)$ at least one witness label of $Y(\ell, j)$ is active during $E(j+1)$. Note that without the explicit constraints (5) and (6) two conflicting labels could switch activity at any point during the conflict interval rather than only at the endpoints.

Theorem 2. *Given an instance $I = (P, L, \Psi, C)$, the ILP (1)–(6) computes an optimal solution Φ of* GENERALMAXTOTAL *in AM3. It uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

We can adapt the above ILP to AM1 and AM2 as follows. For AM2 we replace the right hand side of constraint (5) by 0, and for AM1 we also replace the right hand side of constraint (6) by 0. This excludes exactly the start- and endpoints of the activity intervals that are forbidden in AM1 or AM2. It is easy to see that these ILP formulations can be modified further to solve k-RESTRICTEDMAXTOTAL by adding the constraint $\sum_{\ell \in L} x_i^\ell \leq k$ for each atomic segment $E(i)$.

Corollary 1. *Given an instance $I = (P, L, \Psi, C)$,* GENERALMAXTOTAL *and k-*RESTRICTEDMAXTOTAL *can be solved in AM1, AM2, and AM3 by an ILP that uses $O(N \cdot (|\Psi| + |C|))$ variables and constraints.*

Experiments. We have evaluated the ILP in all three models using Open Street Map data of the city center of Karlsruhe (Germany) which contains more than 2,000 labels. To this end we generated 1,000 shortest paths on the road network of Karlsruhe by selecting source and target vertices uniformly at random and transformed those shortest paths into trajectories consisting of circular arcs. The experimental evaluation in the full version [7] indicates that the ILP formulations are indeed applicable in practice.

3.2 Approximation of GENERALMAXTOTAL

In this section we describe a simple greedy algorithm for GENERALMAXTOTAL in all three activity models assuming that all labels are unit squares anchored at their lower-left corner. Further, we assume that the weight of each presence interval $[a, b]_\ell$ is its length $w([a, b]_\ell) = b - a$.

Starting with an empty solution Φ, our algorithm GREEDYMAXTOTAL removes the longest interval I from Ψ and adds it to Φ, i.e., I is set active. Then, depending on the activity model, it updates all presence intervals that have a conflict with I in Ψ and continues until the set Ψ is empty.

For AM1 the update process simply removes all presence intervals from Ψ that are in conflict with the newly selected interval I. For AM2 and AM3 let $I_j \in \Psi$ and let $I_j^1, \ldots, I_j^k$ be the longest disjoint sub-intervals of I_j that are not in conflict with the selected interval I. We assume that $I_j^1, \ldots, I_j^k$ are sorted by their left endpoint. The update operation for AM2 replaces every interval $I_j \in \Psi$ that is in conflict with I with I_j^1. In AM3 we replace I_j by I_j^1, if I_j^1 is not fully contained in I. Otherwise, I_j is replaced by I_j^k. Note that this discards some candidate intervals, but the chosen replacement of I_j is enough to prove the approximation factor. Note that after each update all intervals in Ψ are valid choices according to the specific model. Hence, we can conclude that the result Φ of GREEDYMAXTOTAL is also valid in that model.

In the following we analyze the approximation quality of GREEDYMAXTOTAL. To that end we first introduce a purely geometric packing lemma. Similar packing lemmas have been introduced before, but to the best of our knowledge for none of them it is sufficient that only one prescribed corner of the packed objects lies within the container.

Lemma 1. *Let C be a circle of radius $\sqrt{2}$ in the plane and let Q be a set of non-intersecting closed and axis-parallel unit squares with their bottom-left corner in C. Then Q cannot contain more than eight squares.*

Based on Lemma 1 we now show that for any label with anchor p there is no point of time $t \in [0, 1]$ for which there can be more than eight active labels whose anchors are within distance $\sqrt{2}$ of p. We call a set $X \subseteq \Psi$ *conflict-free* if it contains no pair of presence intervals that are in conflict. Further, we say that X is in conflict with $I \in \Psi$ if every element of X is in conflict with I, and we say that X contains $t \in [0, 1]$ if every element of X contains t.

Lemma 2. *For every $t \in [0, 1]$ and every $I \in \Psi$ any maximum cardinality conflict-free set $X_I(t) \subseteq \Psi$ that is in conflict with I and contains t satisfies $|X_I(t)| \leq 8$.*

With this lemma we can finally obtain the approximation guarantees for GREEDYMAXTOTAL for all activity models.

Theorem 3. *Assuming that all labels are unit squares and $w([a, b]) = b - a$,* GREEDYMAXTOTAL *is a 1/24-, 1/16-, 1/8-approximation for AM1–AM3, respectively, and needs $O(n \log n)$ time for AM1 and $O(n^2)$ time for AM2 and AM3.*

Proof. To show the approximation ratios, we consider an arbitrary step of GREEDYMAXTOTAL in which the presence interval $I = [a, b]_\ell$ is selected from Ψ. Let C_ℓ^I be the set of presence intervals in Ψ that are in conflict with I.

Consider the model AM1. Since I is the longest interval in Ψ when it is chosen, the intervals in C_ℓ^I must be completely contained in $J = [a - w(I), b + w(I)]$. As C_ℓ^I contains all presence intervals that are in conflict with I it is sufficient to consider J to bound the effect of selecting I. Obviously, the interval J is three times as long as I. By Lemma 2 we know that for any $X_I(t)$ it holds that $|X_I(t)| \leq 8$ for all $t \in J$. Hence, in an optimal solution there can be at most eight active labels at each point $t \in J$ that are discarded when $[a, b]_\ell$ is selected. Thus, the cost of selecting $[a, b]_\ell$ is at most $3 \cdot 8 \cdot w(I)$.

For AM2 we apply the same arguments, but restrict the interval J to $J = [a, b + w(I)]$, which is only twice as long as I. To see that consider for an interval $[c, d]_{\ell'} \in C_\ell^I$ the prefix $[c, a]$ if it exists. If $[c, a]$ does not exist (because $a < c$), removing $[c, d]_{\ell'}$ from Ψ changes Ψ only in the range of J. If $[c, a]$ exists, then again Ψ is only changed in the range of I, because by definition $[c, d]_{\ell'}$ is shortened to an interval that at least contains $[c, a]$ and is still contained in Ψ. Thus, the cost of selecting I is at most $2 \cdot 8w(I)$.

Analogously, for AM3 we can argue that it is sufficient to consider the interval $J = [a, b]$. By definition of the update operation of GREEDYMAXTOTAL at least the prefix or suffix subinterval of each $[c, d]_{\ell'} \in C_\ell^I$ remains in Ψ that extends beyond I (if such an interval exists). Thus, selecting I influences only the interval J and its cost is at most $8w(I)$. The approximation bounds of $1/24$, $1/16$, and $1/8$ follow immediately.

We use a heap to achieve the time complexity $O(n \log n)$ of GREEDYMAXTOTAL for AM1 since each interval is inserted and removed exactly once. For AM2 and AM3 we use a linear sweep to identify the longest interval contained in Ψ. In each step we need $O(n)$ time to update all intervals in Ψ, and we need a total of $O(n)$ steps. Thus, GREEDYMAXTOTAL needs $O(n^2)$ time in total for AM2 and AM3. □

4 Solving k-RESTRICTEDMAXTOTAL

Corollary 1 showed that k-RESTRICTEDMAXTOTAL can be solved by integer linear programming in all activity models. In this section we prove that unlike GENERAL-MAXTOTAL the problem k-RESTRICTEDMAXTOTAL can actually be solved in polynomial time. We give a detailed description of our algorithm for AM1, and then show how it can be extended to AM2. Note that solving k-RESTRICTEDMAXTOTAL is related to finding a maximum cardinality k-colorable subset of n intervals in interval graphs. This can be done in polynomial time in both n and k [4]. However, we have to consider additional constraints due to conflicts between labels, which makes our problem more difficult. First, we discuss how to solve the case for $k = 1$, then give an algorithm that solves k-RESTRICTEDMAXTOTAL for $k = 2$, and finally extend this result recursively to any constant $k > 2$.

4.1 An Algorithm for 2-RESTRICTEDMAXTOTAL

We start with some definitions before giving the actual algorithm. We assume that the intervals of $\Psi = \{I_1, \dots, I_n\}$ are sorted in non-decreasing order by their left endpoints; ties are broken arbitrarily. First note that for the case that at most one label can be active at any given point in time ($k = 1$), conflicts between labels do not matter. Thus, it is sufficient to find an independent subset of Ψ of maximum weight. This is equivalent to finding a maximum weight independent set on interval graphs, which can be done in $O(n)$ time using dynamic programming given n sorted intervals [9]. We denote this algorithm by $\mathcal{A}_1$. Let $L_1[I_j]$ be the set of intervals that lie completely to the left of the left endpoint of I_j. Algorithm $\mathcal{A}_1$ basically computes a table $\mathcal{T}_1$ indexed by the intervals in Ψ, where an entry $\mathcal{T}_1[I_j]$ stores the value of a maximum weight independent set Q of $L_1[I_j]$ and a pointer to the rightmost interval in Q.

We call a pair of presence intervals (I_i, I_j), $i < j$, a *separating pair* if I_i and I_j overlap and are not in conflict with each other. Further, a separating pair $\boldsymbol{v} = (I_p, I_q)$ is *smaller* than another separating pair $\boldsymbol{w} = (I_i, I_j)$ if and only if $p < i$ or $p = i$ and $q < j$. This induces a total order and we denote the ordered set of all separating pairs by $S_2 = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_z\}$. The weight of a separating pair $\boldsymbol{v}$ is defined as $w(\boldsymbol{v}) = \sum_{I \in \boldsymbol{v}} w(I)$.

We observe that a separating pair $\boldsymbol{v} = (I_i, I_j)$ contained in a solution of 2-RESTRICTEDMAXTOTAL splits the set of presence intervals into two independent subsets. Specifically, a left (right) subset $L_2[\boldsymbol{v}]$ ($R_2[\boldsymbol{v}]$) that contains only intervals which lie completely to the left (right) of the intersection of I_i and I_j and are neither in conflict with I_i nor I_j; see Fig. 3.

We are now ready to describe our dynamic programming algorithm $\mathcal{A}_2$. For ease of notation we add two dummy separating pairs to S_2. One pair $\boldsymbol{v}_0$ with presence intervals strictly to the left of 0 and one pair $\boldsymbol{v}_{z+1}$ with presence intervals strictly to the right of 1. Since all original presence intervals are completely contained in $[0, 1]$ every optimal solution contains both dummy separating pairs. Our algorithm computes a one-dimensional table $\mathcal{T}_2$, where for each separating pair $\boldsymbol{v}$ there is an entry $\mathcal{T}_2[\boldsymbol{v}]$ that stores the value of the optimal solution for $L_2[\boldsymbol{v}]$. We compute $\mathcal{T}_2$ from left to right starting with the dummy separating pair $\boldsymbol{v}_0$ and initialize $\mathcal{T}_2[\boldsymbol{v}_0] = 0$. Then, we recursively define $\mathcal{T}_2[\boldsymbol{v}_j]$ for every $\boldsymbol{v}_j \in S_2$ as $\mathcal{T}_2[\boldsymbol{v}_j] = \max_{i<j}\{\mathcal{T}_2[\boldsymbol{v}_i] + w(\boldsymbol{v}_i) + \mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j) \mid \boldsymbol{v}_i \in$

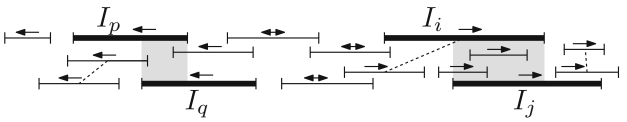


Fig. 3. Illustration of presence intervals. Intervals that are in conflict are connected by a dotted line. Both (I_i, I_j) and (I_p, I_q) are separating pairs. The intervals of $L_2[i, j]$ ($R_2[p, q]$) are marked by a left (right) arrow.

S_2, $\boldsymbol{v}_i \subseteq L_2[\boldsymbol{v}_j]$, $\boldsymbol{v}_j \subseteq R_2[\boldsymbol{v}_i]\}$. Additionally, we store a backtracking pointer to the predecessor pair that yields the maximum value. In other words, for computing $\mathcal{T}_2[\boldsymbol{v}_j]$ we consider all possible direct predecessors $\boldsymbol{v}_i \in S_2$ with $i < j$, $\boldsymbol{v}_i \cap \boldsymbol{v}_j = \emptyset$, and no conflict with $\boldsymbol{v}_j$. Each such $\boldsymbol{v}_i$ induces a candidate solution whose value is composed of $\mathcal{T}_2[\boldsymbol{v}_i]$, $w(\boldsymbol{v}_i)$, and the value of an optimal solution of algorithm $\mathcal{A}_1$ for the intervals between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ with $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ active.

Since by construction $L_2[\boldsymbol{v}_{z+1}] = \Psi \cup \boldsymbol{v}_0$, the optimal solution to 2-RESTRICTEDMAXTOTAL is stored in $\mathcal{T}_2[\boldsymbol{v}_{z+1}]$ once $\boldsymbol{v}_0$ is removed. To compute a single entry $\mathcal{T}_2[\boldsymbol{v}_j]$ our algorithm needs to consider all possible separating pairs preceding $\boldsymbol{v}_j$, and for each of them obtain the optimal solution from algorithm $\mathcal{A}_1$ under some additional constraints. For the call $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$ in the recursive equation above, we distinguish two cases. If the rightmost endpoint of $\boldsymbol{v}_i$ is to the left of the leftmost endpoint of $\boldsymbol{v}_j$ then we run algorithm $\mathcal{A}_1$ on the set of intervals $L_2[\boldsymbol{v}_j] \cap R_2[\boldsymbol{v}_i]$ and obtain the value $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$. Otherwise, there is an overlap between an interval I_a of $\boldsymbol{v}_i$ and an interval I_b of $\boldsymbol{v}_j$. Since for $k = 2$ no other interval can cross this overlap, we actually make two calls to $\mathcal{A}_1$, once on the set $R_2[\boldsymbol{v}_i] \cap L_2[(I_a, I_b)]$ and once on the set $R_2[(I_a, I_b)] \cap L_2[\boldsymbol{v}_j]$. We add both values to obtain $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$. Since we run algorithm $\mathcal{A}_1$ for each of $O(z)$ separating pairs, the time complexity to compute a single entry of $\mathcal{T}_2$ is $O(nz)$. To compute the whole table the algorithm repeats this step $O(z)$ times, which yields a total time complexity of $O(nz^2)$. Note that the number of separating pairs z is in $O(n^2)$.

We prove the correctness of the algorithm by contradiction. Assume that there exists an instance for which our algorithm does not compute an optimal solution and let OPT be an optimal solution. This means, that there is a smallest separating pair $\boldsymbol{v}_j$ for which the entry in $\mathcal{T}_2[\boldsymbol{v}_j]$ is less than the value of OPT for $L_2[\boldsymbol{v}_j]$. Note that $\boldsymbol{v}_j$ cannot be the dummy separating pair $\boldsymbol{v}_0$ since $\mathcal{T}_2[\boldsymbol{v}_0]$ is trivially correct. Let $\boldsymbol{v}_i$ be the rightmost separating pair in OPT that precedes $\boldsymbol{v}_j$ and is disjoint from it (possibly $\boldsymbol{v}_i = \boldsymbol{v}_0$). Since there is no other disjoint separating pair between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ in OPT, all intervals in OPT between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ form a subset of $R_2[\boldsymbol{v}_i] \cap L_2[\boldsymbol{v}_j]$ that is a valid configuration for $k = 1$. We can obtain an optimal solution for $k = 1$ of the intervals in $R_2[\boldsymbol{v}_i] \cap L_2[\boldsymbol{v}_j]$ by computing $\mathcal{A}_1(\boldsymbol{v}_i, \boldsymbol{v}_j)$ as described above. Since, by assumption, $\mathcal{T}_2[\boldsymbol{v}_i]$ is optimal, $\mathcal{A}_1$ is correct [9], and our algorithm explicitly considers all possible preceding separating pairs including $\boldsymbol{v}_i$, the entry $\mathcal{T}_2[\boldsymbol{v}_j]$ must be at least as good as OPT for $L[\boldsymbol{v}_j]$. This is a contradiction and the correctness of $\mathcal{A}_2$ follows.

Theorem 4. *Algorithm $\mathcal{A}_2$ solves* 2-RESTRICTEDMAXTOTAL *in AM1 in $O(nz^2)$ time and $O(z)$ space, where z is the number of separating pairs in the input instance.*

4.2 An Algorithm for k-RESTRICTEDMAXTOTAL

In the following we extend the dynamic programming algorithm $\mathcal{A}_2$ to a general algorithm $\mathcal{A}_k$ for the case $k > 2$. To this end, we extend the definition of separating pairs to separating k-tuples. A *separating k-tuple $\boldsymbol{v}$* is a set of k presence intervals that are not in conflict with each other and that have a non-empty intersection $Y_{\boldsymbol{v}} = \bigcap_{I \in \boldsymbol{v}} I$. We say a separating k-tuple $\boldsymbol{v}$ is *smaller* than a separating k-tuple $\boldsymbol{w}$ if $Y_{\boldsymbol{v}}$ begins to the left of $Y_{\boldsymbol{w}}$. Ties are broken arbitrarily. This lets us define the ordered set $S_k = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_z\}$ of all separating k-tuples of a given set of presence intervals. We say a set C of presence intervals is *k-compatible* if no more than k intervals in C intersect at any point and there are no conflicts in C. Two separating k-tuples $\boldsymbol{v}$ and $\boldsymbol{w}$ are *k-compatible* if they are disjoint and $\boldsymbol{v} \cup \boldsymbol{w}$ is k-compatible. The definitions of the sets $R_2[\boldsymbol{v}]$ and $L_2[\boldsymbol{v}]$ extend naturally to the sets $R_k[\boldsymbol{v}]$ and $L_k[\boldsymbol{v}]$ of all intervals completely to the right (left) of $Y_{\boldsymbol{v}}$ and not in conflict with any interval in $\boldsymbol{v}$. Now, we recursively define the algorithm $\mathcal{A}_k$ that solves k-RESTRICTEDMAXTOTAL given a pair of active k-compatible boundary k-tuples. Note that in the recursive definition these boundary tuples may remain k-dimensional even in $\mathcal{A}_{k'}$ for $k' < k$. For $\mathcal{A}_k$ we define as boundary tuples two k-compatible dummy separating k-tuples $\boldsymbol{v}_0$ and $\boldsymbol{v}_{z+1}$ with all presence intervals strictly to the left of 0 and to the right of 1, respectively. The algorithm fills a one-dimensional table $\mathcal{T}_k$. Similarly to the case $k = 2$, each entry $\mathcal{T}_k[\boldsymbol{v}]$ stores the value of the optimal solution for $L_k[\boldsymbol{v}]$, i.e., the final solution can again be obtained from $\mathcal{T}_k[\boldsymbol{v}_{z+1}]$. We initialize $\mathcal{T}_k[\boldsymbol{v}_0] = 0$. Then, the remaining entries of $\mathcal{T}_k$ can be obtained by computing $\mathcal{T}_k[\boldsymbol{v}_j] = \max_{i<j}\{\mathcal{T}_k[\boldsymbol{v}_i] + w(\boldsymbol{v}_i) + \mathcal{A}_{k-1}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{v}}_j) \mid \boldsymbol{v}_i \in S_k,\, \boldsymbol{v}_i \subseteq L_k[\boldsymbol{v}_j] \cup \boldsymbol{v}_0,\, \boldsymbol{v}_j \subseteq R_k[\boldsymbol{v}_i] \cup \boldsymbol{v}_{z+1},\, \boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1} \cup \boldsymbol{v}_i \cup \boldsymbol{v}_j$ is k-compatible$\}$, which uses the algorithm $\mathcal{A}_{k-1}$ recursively on a suitable subset of presence intervals between the boundary tuples $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$. Here $\tilde{\boldsymbol{v}}_i$ is defined as the union of the tuple $\boldsymbol{v}_i$ and all intervals in $\boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1}$ that intersect the right endpoint of $Y_{\boldsymbol{v}_i}$; analogously $\tilde{\boldsymbol{v}}_j$ is defined as the union of the tuple $\boldsymbol{v}_j$ and all intervals in $\boldsymbol{v}_0 \cup \boldsymbol{v}_{z+1}$ that intersect the left endpoint of $Y_{\boldsymbol{v}_i}$. This makes sure that in each subinstance all active intervals that are relevant for that particular subinstance are known. Note that by the k-compatibility condition $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$ contain at most k elements each. In fact, $\mathcal{A}_{k-1}(\tilde{\boldsymbol{v}}_i, \tilde{\boldsymbol{v}}_j)$ uses $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_j$ as boundary k-tuples (and thus does not create dummy boundary tuples) and the set $R_k[\boldsymbol{v}_i] \cap L_k[\boldsymbol{v}_j]$ as the set of presence intervals from which separating $(k-1)$-tuples can be formed.

Theorem 5. *Algorithm $\mathcal{A}_k$ solves k-RESTRICTEDMAXTOTAL in AM1 in $O(n^{k^2+k-1})$ time and $O(n^k)$ space.*

It is natural to ask whether it is possible to extend the dynamic program described above to the models AM2 and AM3. With some modifications and at the expense of another polynomial factor in the running time we can extend algorithm $\mathcal{A}_k$ to the activity model AM2. The important difference between AM1 and AM2 is that presence intervals can be truncated at their right side if there is an active conflicting witness label causing the truncation. Hence, we need to add for each presence interval, all possible subintervals to Ψ that might be contained in an optimal solution. Moreover special care needs to be taken to ensure the witness condition of AM2 for all truncated intervals. A more detailed discussion of this extension can be found in the full version [7].

Theorem 6. k-RESTRICTEDMAXTOTAL *in AM2 can be solved in polynomial time.*

It remains open whether k-RESTRICTEDMAXTOTAL can be solved in polynomial time in AM3. Another extension of the dynamic programming algorithm is unlikely, since in AM3 the left and right subinstances created by a separating k-tuple $\boldsymbol{v}$ may have dependencies and thus cannot be solved independently any more. This is because a single original presence interval I can have subintervals both in $L_k[\boldsymbol{v}]$ and $R_k[\boldsymbol{v}]$, which cannot simultaneously be active.

Since the running time of our algorithms are, even for small k, prohibitively expensive in practice, we propose an approximation algorithm for k-RESTRICTEDMAXTOTAL based on GREEDYMAXTOTAL, which can be found in the full version [7].

Theorem 7. *There exists an $O(n^2)$-time approximation algorithm for for* k-RESTRICTEDMAXTOTAL *with unit squares for AM1–AM3 with approximation ratios* $1/\min\{3+3k, 27\}$, $1/\min\{3+2k, 19\}$, $1/\min\{3+k, 11\}$, *respectively.*

Acknowledgment. This work was supported by the 'Concept for the Future' of KIT under grant YIG 10-209 and by a Google Research Award.

References

1. Agarwal, P.K., van Kreveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Comput. Geom. Theory & Appl. 11(3-4), 209–218 (1998)
2. Been, K., Daiches, E., Yap, C.: Dynamic map labeling. IEEE Trans. Visualization and Computer Graphics 12(5), 773–780 (2006)
3. Been, K., Nöllenburg, M., Poon, S.-H., Wolff, A.: Optimizing active ranges for consistent dynamic map labeling. Comput. Geom. Theory & Appl. 43(3), 312–328 (2010)
4. Carlisle, M.C., Lloyd, E.L.: On the k-coloring of intervals. Discr. Appl. Math. 59(3), 225–235 (1995)
5. Chalermsook, P., Chuzhoy, J.: Maximum independent set of rectangles. In: ACM-SIAM Symp. Discr. Algorithms (SODA 2009), pp. 892–901 (2009)
6. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inform. Process. Lett. 12(3), 133–137 (1981)
7. Gemsa, A., Niedermann, B., Nöllenburg, M.: Trajectory-based dynamic map labeling. CoRR, arXiv:1309.3963 (2013)
8. Gemsa, A., Nöllenburg, M., Rutter, I.: Consistent labeling of rotating maps. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 451–462. Springer, Heidelberg (2011)
9. Hsiao, J.Y., Tang, C.Y., Chang, R.S.: An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. Inform. Process. Lett. 43(5), 229–235 (1992)
10. Marx, D.: Efficient approximation schemes for geometric problems? In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 448–459. Springer, Heidelberg (2005)
11. Niedermann, B.: Consistent labeling of dynamic maps using smooth trajectories. Master's thesis, Karlsruhe Institute of Technology (June 2012)
12. Sester, M., Brenner, C.: Continuous generalization for visualization on small mobile devices. In: Fisher, P.F. (ed.) Spatial Data Handling (SDH 2004), pp. 355–368. Springer (2004)
13. Wagner, F., Wolff, A.: A practical map labeling algorithm. Comput. Geom. Theory Appl. 7, 387–404 (1997)
14. Wagner, F., Wolff, A., Kapoor, V., Strijk, T.: Three rules suffice for good label placement. Algorithmica 30, 334–349 (2001)

Asynchronous Rumor Spreading on Random Graphs

Konstantinos Panagiotou and Leo Speidel

University of Munich, Mathematics Institute,
Theresienstr. 39, 80333 Munich, Germany
kpanagio@math.lmu.de, leo.speidel@outlook.com

Abstract. We perform a thorough study of various characteristics of the asynchronous push-pull protocol for spreading a rumor on Erdős-Rényi random graphs $G_{n,p}$, for any $p > c\ln(n)/n$, $c > 1$. In particular, we prove tight bounds for the total time that is needed until the information has spread to all nodes. Moreover, we quantify precisely the robustness of the protocol with respect to transmission and node failures.

1 Introduction

Rumor spreading protocols have become fundamental mechanisms for designing efficient and fault-tolerant algorithms that disseminate information in large and complex networks. In the classical setting the algorithm that we will consider proceeds in synchronous rounds. Initially, some arbitrary node receives a piece of information. In each subsequent round, every node that knows the information transmits it to a randomly selected neighbor in the network. This operation is denoted as a *push*. Moreover, every node that does not possess the information tries to learn it from a randomly selected neighbor; this operation is denoted as a *pull*. Equivalently, we can say that in every round, every node contacts a randomly chosen neighbor and exchanges the information with it.

Rumor spreading was first introduced in [9], where the problem of distributing updates consistently in replicated databases was considered. Subsequently it has found many other applications, such as the detection of failures in a distributed environment [22], sampling of peers [16] and averaging in networks that consist of many sensors in a distributed fashion [3].

In this work we consider a variation of the classical push-pull algorithm that was introduced in [4]. In that paper, striving for a more realistic setting, the authors modified the algorithm by dropping the assumption that all nodes are able to act in synchrony. In the *asynchronous version* that we consider here, nodes do not contact other nodes simultaneously in synchronized rounds, but do so in times that arrive according to independent rate 1 Poisson processes at each node. In [4] this is suggested as a possible solution if a centralized entity for facilitating time synchronization is not existent or has failed in the networks that we consider.

1.1 Results

In this paper we present a thorough study of various characteristics of the asynchronous push-pull algorithm. We will assume that the underlying network is an Erdős-Renyi random graph $G_{n,p}$, where each edge is included independently of all other edges with probability p. For any $p > c\ln(n)/n$, $c > 1$, we show almost optimal bounds for the time that is needed until the information has spread to all nodes. We also precisely quantify the robustness of the algorithm with respect to transmission and node failures.

Let us introduce some basic notation first. For a graph G with n nodes we assume that its node set is $[n]$, where $[n]$ denotes the set $\{1, \dots, n\}$. For $v \in [n]$ we write $N_G(v)$ for the neighborhood of v and $d_G(v) = |N_G(v)|$. For any $S \subseteq [n]$ we abbreviate $N_G(S) = \cup_{v\in S} N_G(v)$. Moreover, we denote by $e_G(S,R)$ the number of edges between $S, R \subseteq [n]$ and abbreviate this quantity by $e_G(S)$ if $S = R$. If G is clearly given from the context we may drop the subscript G in our notation. Finally, we let $T(G)$ denote the (random) time that the asynchronous push-pull protocol needs to spread a rumor to all nodes in that graph. Note that when studying rumor spreading on random graphs, we effectively have two probability spaces - one for the random graph and one for rumor spreading.[1]

Our first result addresses the performance of the algorithm on random graphs with an edge probability that is significantly above the connectivity threshold.

Theorem 1. *Let $p = \alpha(n)\ln(n)/n$ for some $\alpha(n) = \omega(1)$. Then w.h.p.*

$$\mathbb{E}[T(G_{n,p})] = \left(1 \pm \sqrt{34/\alpha(n)}\right) H_{n-1} + O\left(\ln(n)/n\right) \text{ and}$$
$$T(G_{n,p}) = \mathbb{E}[T(G_{n,p})] + O(\alpha(n)^{-1/2}\ln(n) + 1).$$

Some remarks are in place. First of all, note that if $p \geq \ln^3(n)/n$, then the theorem states that w.h.p. $\mathbb{E}[T(G_{n,p})] = H_{n-1} + O(1)$, and further that w.h.p. $T(G_{n,p}) = \mathbb{E}[T(G_{n,p})] + O(1)$. This is a very tight result, and indeed it is best possible, since just the time until the second node is informed has a variance of $\Omega(1)$. For almost no other rumor spreading protocol such a precise result for the total time is known (see Section 1.2 for a discussion). On the other hand, this result might not be completely unexpected: It is not very difficult to see that for the complete graph K_n, $T(K_n) = H_{n-1} + O(1)$ w.h.p. However, for $p = \omega(\ln(n)/n)$, the edges of $G_{n,p}$ are spread very uniformly, in the sense that between *any* two sets of nodes, the number of edges is close to the expected value. So, the dynamics of the information spreading process are not affected crucially by the fact that the graph does not contain all edges.

For p closer to the connectivity threshold the edges are not distributed as uniformly as in the former case and a different behavior might be expected.

[1] Some additional notation will be used as well: We write H_n for $\sum_{j=1}^{n} \frac{1}{j}$, $\ln(n)$ for the natural logarithm and $\log_b(n) = \ln(n)/\ln(b)$ for any $b > 1$. For any $a, b \in \mathbb{R}$ we write $a \pm b$ for the interval $(a-b, a+b)$ and abbreviate $X = a \pm b$ for $X \in (a-b, a+b)$. W.h.p. abbreviates "with high probability" and means that an event (dependent on n) occurs with probability $1 - o(1)$ as $n \to \infty$.

Indeed it has been observed for many rumor spreading protocols that these fluctuations may have a significant impact on the time needed to inform all nodes (see Section 1.2 for a discussion). Our second result, however, shows that the time required by the asynchronous push-pull protocol is *asymptotically* unaffected by the average degree of the graph.

Theorem 2. *Let $p = c\ln(n)/n$ for some $c > 1$. Then w.h.p.*

$$\mathbb{E}[T(G_{n,p})] = \ln(n) + O(\ln^{3/4}(n)) \text{ and } T(G_{n,p}) = \mathbb{E}[T(G_{n,p})] + O(\ln^{3/4}(n)).$$

This result quantifies an important property of the asynchronous push-pull algorithm that has not been studied so precisely in previous works: the robustness of the algorithm with respect to the distribution of the edges and the average degree of the underlying networks. Indeed, for random graphs the total time needed is asymptotically not affected at all by these parameters.

We also study the robustness of the algorithm with respect to other parameters. First, suppose that every time a node contacts some other node the connection is dropped independently of other connections with probability $1 - q$, where $0 < q \leq 1$, before any information can be exchanged. Let $T_q(G)$ be the time until all nodes receive the information. Our next result quantifies the effect of the "success probability" q on the total time.

Theorem 3. *Let $p > \ln(n)/n$. Then w.h.p. $\mathbb{E}[T_q(G_{n,p})] = 1/q\ \mathbb{E}[T(G_{n,p})]$. Moreover, $T(G_{n,p})$ is bounded around its expected value asymptotically as accurate as in Theorem 1 and 2.*

Note, that $1/q$ is exactly the expected number of connection attempts that have to be made until the connection is not dropped for the first time. Therefore, this result also demonstrates the robustness and the adaptivity of the asynchronous push-pull algorithm that essentially slows down at the least possible rate. In many other rumor spreading protocols the effect of q on the total time is larger.

Finally, we study the robustness with respect to node failures. Suppose that in the given network a random subset B of nodes is declared "faulty" in the sense that even if they receive the rumor, they will neither perform any push operation, nor will they respond to any pull request. Let $T_B(G)$ denote the time until the information has spread to all nodes. Our next result states that $T_B(G_{n,p})$ is asymptotically equal to $T(G_{n,p})$, provided that B is not linear in n.

Theorem 4. *Let p be as in Theorem 1 or 2, and suppose that $B = o(n)$. Then w.h.p. $T_B(G_{n,p}) = (1 + o(1))\mathbb{E}[T(G_{n,p})]$.*

1.2 Related Work

There are many theoretical studies that are concerned with the performance of the *synchronous* push-pull algorithm [5,6,10,13,15,17]. For example, the performance of rumor spreading on general graph topologies was made explicit in [5,6,15], where the number of rounds necessary to spread a rumor was related to the conductance of the graph. In particular, the upper bound $O(\varphi^{-1}\ln(n))$ was shown, where φ is the conductance of the graph.

Very accurate analyses exist also for the synchronous push protocol, i.e., where only push transmissions are allowed [12,14,20,21]. In [21] it was shown that a rumor spreads in $\log_2(n) + \ln(n) + O(1)$ rounds w.h.p. on the complete graph. However, in contrast to what we prove in this work, tight bounds on the spreading times are not even known for dense random graphs. In [12] it was shown that for $G_{n,p}$ with $p = \omega(\ln(n)/n)$ a rumor spreads w.h.p. in $\log_2(n) + \ln(n) + o(\ln(n))$ rounds. Furthermore, it was shown in [20] that for $G_{n,p}$ with $p = c\ln(n)/n$, $c > 1$, the spreading time increases significantly to $\log_2(n)+\gamma(c)\ln(n)+o(\ln(n))$ w.h.p., where $\gamma(c) = c\ln(c/(c-1))$. This means in particular that the edge probability has a direct impact on the spreading time, which is in contrast to the asynchronous push-pull algorithm that is robust with respect to variations in the average degree, cf. Theorem 2.

The effect of transmission failures that occur independently for each contact at some constant rate $1-q$ was investigated in [12] for the synchronous push protocol on dense random graphs. There, it was shown that the total time increases over-proportionally, namely by a factor larger than q^{-1}.

For the asynchronous push-pull protocol there exists much less literature [11,13], mostly devoted to models for social networks. In [11] it was shown that on preferential attachment graphs [2] a rumor needs a time of $O(\sqrt{\ln(n)})$ w.h.p. to spread to almost all nodes. On power-law Chung-Lu random graphs [7] (for $2 < \beta < 3$) it was shown in [13] that a rumor initially located within the giant component spreads w.h.p. even in constant time to almost all nodes.

2 Analysis of the Protocol

In this section we describe a general strategy for analyzing the spreading time on arbitrary graphs. We will apply this strategy to $G_{n,p}$ in the section hereafter. Let G be any connected graph with n nodes. Towards studying the distribution of $T(G)$ we divide the rumor spreading process into n *states*, where in state $1 \le j \le n$, j nodes are informed. We denote the set of informed nodes in state j by $I_j = I_j(G)$ and the set of uninformed nodes in state j by $U_j = U_j(G) = [n] \setminus I_j(G)$. We denote by $t_j = t_j(G)$ the (random) time that the protocol needs to move from state j to $j+1$, where $1 \le j < n$. Clearly $T(G) = \sum_{j=1}^{n-1} t_j$.

Assume that $1 \le j < n$ nodes are informed. We provide a master lemma that determines the distribution of t_j for an arbitrary set of informed nodes.

Lemma 1. *Let $1 \le j < n$. Then t_j is exponentially distributed with parameter*

$$Q_j := \sum_{v\in I_j} |N(v)\cap U_j|/d(v) + \sum_{w\in U_j} |N(w)\cap I_j|/d(w).$$

Moreover, conditional on I_j the time t_j is independent of $t_1, \dots t_{j-1}$.

Proof. We assume that the times $t_1, \dots, t_{j-1}$ and I_j are known and determine the distribution of t_j. In particular, we show that t_j is independent of the times $t_1, \dots, t_{j-1}$. For technical reasons we will align the Poisson processes at certain points in the proof. We note that the memorylessness property of the exponential distribution allows to restart all Poisson processes all over again at any time.

Assume that the Poisson process of $v \in I_j$ has ticked. Then the probability that v informs an uninformed node is $|N(v) \cap U_j|/d(v)$. Similarly, if the Poisson process of $w \in U_j$ has ticked, the probability that w pulls the message from an informed node equals $|N(w) \cap I_j|/d(w)$. Since all nodes are equipped with rate 1 Poisson processes it is equally possible for any Poisson process to tick for the first (next) time. Therefore at any tick the probability that an uninformed node gets informed is: $q_j = 1/n \left(\sum_{v \in I_j} |N(v) \cap U_j|/d(v) + \sum_{w \in U_j} |N(w) \cap I_j|/d(w) \right)$. It follows that the number of ticks σ until an uninformed node gets the message is geometrically distributed with parameter q_j, and it is independent of the actual times between any two consecutive ticks.

Let $\{\tau_\ell\}_{\ell=1}^{\infty}$ be the time between any two consecutive ticks. If we restart all Poisson processes after we observed a tick, then clearly τ_ℓ is the minimum of the individual waiting times of the Poisson processes. These waiting times are independently and exponentially distributed with parameter 1. Since the minimum of independently and exponentially distributed random variables is again exponentially distributed with the sum of the individual parameters we get that $\tau_\ell \sim \mathsf{Exp}(n)$. Moreover, $\{\tau_\ell\}_{\ell=1}^{\infty}$ are independent because the properties of the Poisson processes guarantee that the waiting times after a restart are independent from the waiting times prior to a restart and have the same distribution. Using these considerations we may write $t_j = \sum_{\ell=1}^{\sigma} \tau_\ell$, where $\sigma \sim \mathsf{Geo}(q_j)$, $\tau_\ell \sim \mathsf{Exp}(n)$ and all these random variables are independent. To complete the proof it is therefore sufficient to argue that t_j is exponentially distributed with parameter nq_j. To this end we consider the characteristic function of t_j and make use of the well known one to one correspondence between characteristic functions and cumulative distribution functions (discussed e.g. in [18]). We get $\mathbb{E}\left[e^{ixt_j}\right] = nq_j/(nq_j - ix)$, which is the characteristic function of an exponentially distributed random variable with parameter $Q_j = nq_j$. □

3 The Expected Spreading Time on Random Graphs

Here we apply the results of the previous section to $G_{n,p}$. In particular, we determine the expected value of $T(G_{n,p})$ thus proving the first statements of Theorem 1 and 2 respectively.

3.1 The Case $p = \omega(\ln(n)/n)$

Our goal is to compute the expectation of $T(G_{n,p})$ for $p = \alpha(n)\ln(n)/n$, $\alpha(n) = \omega(1)$. We will actually show a stronger result, namely that the conclusion of Theorem 1 remains valid even when we replace $G_{n,p}$ with *any* graph G that satisfies the following property: for all $S \subseteq [n]$

$$e(S, [n] \setminus S) = \left(1 \pm \sqrt{8/\alpha(n)}\right) |S| \left(n - |S|\right) p. \qquad (1)$$

Note that for $p = \omega(\ln(n)/n)$, $G_{n,p}$ satisfies this property w.h.p.[2] In words, this means that the edges of $G_{n,p}$ are w.h.p. distributed very "uniformly", in the sense that between any $S \subseteq [n]$ and its complement the number $e(S, [n] \setminus S)$ is very close to its expected value $|S|(n - |S|)p$. We first specify the distribution of $t_j(G)$ by applying Lemma 1.

Corollary 1. *Let G satisfy* (1) *for all $S \subseteq [n]$. If n is sufficiently large, then for any I_j, the distribution of $t_j(G)$ conditional on I_j is an exponential distribution with parameter*

$$\left(1 \pm \sqrt{33/\alpha(n)}\right) 2j(n-j)/(n-1). \tag{2}$$

Proof. Let I_j be any (connected) set with j nodes. For any node v in G, by using (1) with $S = \{v\}$ we infer that $d(v) = \big(1 \pm \sqrt{8/\alpha(n)}\big)(n-1)p$. Moreover, we have $\sum_{v \in I_j} |N(v) \cap U_j| = e(I_j, U_j)$. By applying (1) for a second time we obtain that $\sum_{v \in I_j} |N(v) \cap U_j|/d(v) = \big(1 \pm \sqrt{8/\alpha(n)}\big)/\big(1 \mp \sqrt{8/\alpha(n)}\big)\ j(n-j)/(n-1)$ and simple algebraic transformations imply that we can bound this expression for sufficiently large n by $\big(1 \pm \sqrt{33/\alpha(n)}\big) j(n-j)/(n-1)$. We repeat the above calculation with I_j and U_j interchanged. Finally, we plug this into Q_j of Lemma 1 and the statement is shown. □

We have now everything together to calculate the expectation of $T(G)$. Note that the first statement of Theorem 1 follows immediately from Lemma 2.

Lemma 2. *Let G be as in Corollary 1. Then*

$$\mathbb{E}[T(G)] = \left(1 \pm \sqrt{34/\alpha(n)}\right) H_{n-1} + O\left(\ln(n)/n\right).$$

Proof. Using Corollary 1 we obtain that $\mathbb{E}[T(G)]$ equals

$$\sum_{j=1}^{n-1} \mathbb{E}[t_j] = \sum_{j=1}^{n-1} \mathbb{E}[\mathbb{E}[t_j \mid I_j]] = \big(1 \pm \sqrt{33/\alpha(n)}\big)^{-1} \sum_{j=1}^{n-1} (n-1)/(2j(n-j)).$$

In additionally making use of the bound $(1 \pm \sqrt{33/\alpha(n)})^{-1} = 1 \pm \sqrt{34/\alpha(n)}$ for sufficiently large n and using the identity $1/j(n-j) = 1/jn + 1/n(n-j)$ we obtain that $\mathbb{E}[T(G)] = \left(1 \pm \sqrt{34/\alpha(n)}\right)(H_{n-1} - H_{n-1}/n)$. Together with $H_n = \ln(n) + O(1)$ we finally arrive at the claimed bound. □

3.2 The Case $p = c\ln(n)/n$, $c > 1$

We will again show a stronger result and replace $G_{n,p}$ by some graph G satisfying certain properties that hold w.h.p. However, for the case that $p = c\ln(n)/n$, where $c > 1$, the property that (1) holds for all $S \subseteq [n]$ is w.h.p. false; actually, there are significant fluctuations in the quantity $e(S, [n] \setminus S)$ for different sets S of

[2] A proof can be found e.g. in [12].

the same size. Instead we will exploit the properties described in (3),(4),(5),(7) and (10) below. [3]

We begin with determining the distribution of $t_j(G)$; here the bounds are not as tight as in Corollary 1 for all j, but they will suffice for our purposes.

Corollary 2. *Let $c > 1$ and G be some graph satisfying (3),(4),(5),(7) and (10) below. Then there are $C'(c) > C(c) > 0$ such that for any I_j the distribution of $t_j(G)$ conditional on I_j is an exponential distribution with parameter*

i) in $(C \min\{j, n-j\}, C' \min\{j, n-j\})$ for $1 \le j, n-j \le \ln(n)$,
ii) in $\big(1 \pm C \ln^{-1/4}(n)\big) 2 \min\{j, n-j\}$ for $\ln(n) \le j, n-j \le n/\ln^3(n)$,
iii) in $(C \min\{j, n-j\}, C' \min\{j, n-j\})$ for $n/\ln^3(n) \le j \le n - n/\ln^3(n)$.

Proof. Let I_j be some connected subset of $[n]$ with j elements. We apply Lemma 1 and determine Q_j. Let us begin with the case $j \le \ln(n)$. We assume the following two properties for G:

$$\text{for any } v \in [n], \qquad d(v) = \Theta(\ln(n)), \tag{3}$$

$$\text{for any } |S| \le n/2, \ e(S, [n] \setminus S) = \Theta(|S| \ln(n)). \tag{4}$$

These two properties together imply that Q_j equals

$$\Theta\Big(\Big(\sum_{v \in I_j} |N(v) \cap U_j| + \sum_{w \in U_j} |N(w) \cap I_j|\Big) / \ln(n)\Big) = \Theta\left(e(I_j, U_j)/\ln(n)\right) = \Theta(j).$$

Next we consider the case $\ln(n) \le j \le n/\ln^3(n)$, where we bound Q_j more accurately. We begin with the first sum in the expression for Q_j. Using (3) we obtain $\sum_{v \in I_j} |N(v) \cap U_j|/d(v) = j - \sum_{v \in I_j} |N(v) \cap I_j|/d(v) = j - \Theta\left(e(I_j)/\ln(n)\right)$. To estimate $e(I_j)$ we assume the following for G:

$$\text{For any } S \subseteq [n] \text{ with } |S| \le n/\ln(n), \ e(S) \le |S| \ln\ln(n). \tag{5}$$

This then implies that

$$j - \Theta\left(e(S)/\ln(n)\right) = \left(1 - O\left(\ln\ln(n)/\ln(n)\right)\right) j. \tag{6}$$

We move on with the second sum of Q_j. We split this sum into three parts, namely into $\{w \in N'(I_j)\}$, $\{w \in N(I_j) \setminus N'(I_j)\}$ and $\{w \in U_j \setminus N(I_j)\}$, where we define $N'(S)$ to be the set containing all nodes that are outside of S but belong to the neighborhood of S and have a degree in $c\ln(n) \pm \ln^{3/4}(n)$, where $c\ln(n) = pn$. The reason for doing so is that we may assume the following: For any $v \in [n]$ we let $N'(v) := \{w \in N(v) : d(w) = c\ln(n) \pm \ln^{3/4}(n)\}$. Then

$$\text{for any } v \in [n], |N(v) \setminus N'(v)| \le \ln^{3/4}(n). \tag{7}$$

[3] Due to space limitations we omit the proofs that $G_{n,p}$ for $p = c\ln(n)/n$, $c > 1$, w.h.p. satisfies properties (3), (7) and (10). Proofs that (4) and (5) hold w.h.p. can be found e.g. in [8] (Property 3 and its proof).

Among the three sums, the latter sum equals zero so that

$$\sum_{w \in U_j} |N(w) \cap I_j|/d(w) = \sum_{\substack{v \in N'(I_j), \\ w \in N(I_j) \setminus N'(I_j)}} |N(v) \cap I_j|/d(v) + |N(w) \cap I_j|/d(w).$$

Using (3) and the definition of $N'(I_j)$ we infer that this is

$$e(I_j, N'(I_j))/(pn \pm \ln^{3/4}(n)) + e(I_j, N(I_j) \setminus N'(I_j))/\Theta(\ln(n)). \tag{8}$$

Thus we need to consider $e(I_j, N'(I_j))$ and $e(I_j, N(I_j) \setminus N'(I_j))$. By applying (7) we have that

$$e(I_j, N(I_j) \setminus N'(I_j)) \leq \sum_{v \in I_j} |N(v) \setminus N'(v)| \leq j \ln^{3/4}(n). \tag{9}$$

To estimate $e(I_j, N'(I_j))$ we assume a further property: for any connected $S \subseteq [n]$ with $\ln(n) \leq |S| \leq n/2$ we assume that

$$e(S, [n] \setminus S) = (1 \pm \varepsilon(n))\, |S|(n - |S|)p, \tag{10}$$

where $\varepsilon(n) = (24 \ln\ln(n)/\ln(n))^{1/2}$. Note that I_j is necessarily a connected set in G, as any node gets the rumor from one of its neighbors. Thus applying (9) and (10) we obtain that $e(I_j, N'(I_j)) = \big(1 - O(\ln^{-1/4}(n))\big)j(n-j)p$. Together with (6), (8) and (9) this implies $Q_j = \big(1 - O(\ln^{-1/4}(n))\big)2j$.

Next we consider the case $n/\ln^3(n) \leq j \leq n - n/\ln^3(n)$. Again using that $d(v) = \Theta(\ln(n))$, and that $e(I_j, U_j) = \Theta(\min\{j, n-j\}\ln(n))$ we obtain that $Q_j = \Theta\left(e(I_j, U_j)/\ln(n)\right) = \Theta(\min\{j, n-j\})$. This shows the statement also for the case $n/\ln^3(n) \leq j \leq n - n/\ln^3(n)$. Finally, the claims for $1 \leq n - j \leq \ln(n)$ and $\ln(n) \leq n - j \leq n/\ln^3(n)$ follow analogously by interchanging the roles of I_j and U_j in the above steps. □

We are now able to compute the expectation of $T(G)$. Note that the first statement of Theorem 2 follows immediately from Lemma 3. The proof goes similarly as for Lemma 2 and is omitted due to space limitations.

Lemma 3. *Let $c > 1$ and let G be any graph satisfying* (3),(4),(5),(7) *and* (10). *Then* $\mathbb{E}[T(G)] = \ln(n) + O(\ln^{3/4}(n))$.

4 The Actual Spreading Time on Random Graphs

In this section we will complete the proofs of Theorem 1 and 2 in Corollary 3 and 4 respectively. As in the previous section we prove a stronger statement in replacing $G_{n,p}$ by any graph G that satisfies the assumptions made in Corollary 1 or 2. The proof of Corollary 4 goes analogously as for Corollary 3 and is omitted here.

Corollary 3. *Let G be any graph satisfying the assumption of Corollary 1. Then for any $\lambda \in (0,2)$ we have for n large enough and any $t > 0$ that* $\Pr\left[|T(G) - \mathbb{E}[T(G)]| > t\right] \leq \exp\left\{4\sqrt{34/\alpha(n)}\ln(n) - \lambda t + O\left(1\right)\right\}.$

Proof. We prove the statement separately for the upper tail and the lower tail. We begin with the upper tail. Here, Markov's inequality for the monotonically increasing function $e^{\lambda x}$, $\lambda > 0$ implies

$$\Pr\left[T(G) > \mathbb{E}[T(G)] + t\right] \leq \mathbb{E}\left[e^{\lambda T(G)}\right] e^{-\lambda \mathbb{E}[T(G)] - \lambda t}. \tag{11}$$

Using Corollary 1 we can verify that a sequence $\{t_j^+\}_{j=1}^{n-1}$ of independent random variables, where $t_j^+ \sim \mathsf{Exp}\big((1 - \sqrt{33/\alpha(n)})2j(n-j)/(n-1)\big)$, dominates stochastically $\{t_j\}_{j=1}^{n-1}$. Letting $T^+(G) := \sum_{j=1}^{n-1} t_j^+$ we can thus bound the above expression by

$$\mathbb{E}\left[e^{\lambda T(G)}\right] \leq \mathbb{E}\big[e^{\lambda T^+(G)}\big] \leq \mathbb{E}\big[e^{\lambda T^+(G) + O(1)}\big] = \exp\left\{\lambda \mathbb{E}[T^+(G)] + O(1)\right\}.$$

We omit the proof of the second inequality due to space limitations here. Note that λ has to be restricted to the interval $\big(0, 2\big(1 - \sqrt{33/\alpha(n)}\big)\big)$. Plugging this into (11) and using that $\lambda < 2$ and $H_n = \ln(n) + O(1)$ we arrive at the bound

$$\Pr\left[T(G) > \mathbb{E}[T(G)] + t\right] \leq \exp\left\{4\sqrt{34/\alpha(n)}\ln(n) - \lambda t + O(1)\right\}. \tag{12}$$

For the lower tail we also apply Markov's inequality for $e^{\lambda x}$, $\lambda > 0$ and obtain $\Pr\left(T(G) < \mathbb{E}[T(G)] - t\right) \leq \mathbb{E}\left[e^{-\lambda T(G)}\right] \exp\left\{\lambda \mathbb{E}[T(G)] - \lambda t\right\}$. To bound this expression we use that a sequence $\{t_j^-\}_{j=1}^{n-1}$ of independent random variables, where $t_j^- \sim \mathsf{Exp}\big((1 + \sqrt{33/\alpha(n)})2j(n-j)/(n-1)\big)$, is stochastically dominated by $\{t_j\}_{j=1}^{n-1}$. After the same steps as for the upper tail we deduce that for $\lambda < 2$ we get the same bound as in (12). □

Corollary 4. *Let G be any graph satisfying the assumptions of Corollary 2 and take $C' > 0$ from there. Then for any $\lambda \in (0, C')$ we have for n large enough and any $t > 0$ that* $\Pr\big[|T(G) - \mathbb{E}[T(G)]| > t\big] \leq \exp\big\{-\lambda t + O\big(\ln^{3/4}(n)\big)\big\}.$

5 Variations of the Asynchronous Push-Pull Protocol

5.1 The Effect of Transmission Failures – Theorem 3

In this section we consider a more general version of the asynchronous push-pull protocol, in which nodes succeed to push or pull the rumor with probability $q \in (0, 1]$ and fail to do so with probability $1 - q$ independently of any other contacts established between any two nodes. Our aim is to give a statement in the spirit of Lemma 1 for this "faulty" version. We get the following quantitative statement.

Lemma 4. *Let $1 \leq j < n$. Then t_j is exponentially distributed with parameter $q \cdot Q_j$, where Q_j is given in Lemma 1. Moreover, conditional on I_j the time t_j is independent of $t_1, \dots t_{j-1}$.*

Proof. We only highlight the differences to the prove of Lemma 1. If the Poisson process of an informed node $v \in I_j$ has ticked, the probability that v informs an uninformed node is $q|N(v) \cap U_j|/d(v)$. Similarly, if the Poisson process of an uninformed node $w \in U_j$ has ticked, the probability that w pulls the rumor from an informed node is $q|N(w) \cap I_j|/d(v)$. As in the proof of Lemma 1 we infer that at any tick the probability that an uninformed node gets informed is $q_j(q) = q/n \left(\sum_{v \in I_j} |N(v) \cap U_j|/d(v) + \sum_{w \in U_j} |N(w) \cap I_j|/d(w) \right)$. The rest of the proof is exactly the same as the proof of Lemma 1, where we replace all occurrences of q_j by $q_j(q)$. $\square$

With this statement at hand we can repeat all steps performed in Section 3 and 4 to study the effect of the success probability q on the time that is required to spread the rumor to all nodes of $G_{n,p}$. The steps are literally the same, with the only difference being that the parameters in all involved geometric distributions are multiplied by an additional factor of q. For example, in Corollary 1, Equation (2) is replaced by $\left(1 \pm \sqrt{33/\alpha(n)}\right) 2qj(n-j)/(n-1)$ and similarly, in the conclusions i), iii) and ii) of Lemma 2 we get the bounds $\Theta(q \min\{j, n-j\})$ and $(1 + O(\ln^{-1/4}(n)))2q \min\{j, n-j\}$. In consequence, in the conclusions of Lemmas 2 and 3 the expected values are multiplied by q^{-1}. The proofs in Section 4 are adapted accordingly to obtain asymptotically the same bounds as in Theorem 1 and 2.

5.2 The Effect of Faulty Nodes – Theorem 4

Suppose that before the rumor is spread by the asynchronous push-pull protocol, a random subset of the nodes of $G_{n,p}$ is declared "faulty", in the sense that even if they receive the rumor, they will neither perform any push operation, nor will they respond to any pull request. Note that if the subset of faulty nodes is of size $o(n)$ and $p \geq (1+\varepsilon)\ln(n)/n$ for some $\varepsilon > 0$, then the subgraph of $G_{n,p}$ induced by the non-faulty nodes is distributed like $G_{n',p'}$, where $n' = (1-o(1))n$ and $p' \geq (1+\varepsilon - o(1))\frac{\ln(n)}{n}$. Thus, if the initially informed node does not fault, then the results in the previous sections apply also in this case, where we replace n and p by n' and p' respectively.

References

1. Aigner, M., Ziegler, G.: Proofs from THE BOOK. Springer (2010)
2. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science 286, 509–512 (1999)
3. Boyd, S., Arpita, G., Balaji, P., Devavrat, S.: Gossip algorithms: Design, analysis and applications. In: 24th INFOCOM, pp. 1653–1664. IEEE (2005)

4. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Transactions on Information Theory 52(6), 2508–2530 (2006)
5. Chierichetti, F., Lattanzi, S., Panconesi, A.: Almost tight bounds for rumour spreading with conductance. In: 42nd ACM Symposium on Theory of Computing (STOC 2010), pp. 399–408 (2010)
6. Chierichetti, F., Lattanzi, S., Panconesi, A.: Rumour spreading and graph conductance. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 1657–1663 (2010)
7. Chung, F., Lu, L.: The average distance in a random graph with given expected degrees. Internet Mathematics 1(1), 91–114 (2003)
8. Cooper, C., Frieze, A.: The cover time of sparse random graphs. Random Structures and Algorithms 30(1-2), 1–16 (2007)
9. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (POCD 1987), pp. 1–12 (1987)
10. Doerr, B., Fouz, M., Friedrich, T.: Social networks spread rumors in sublogarithmic time. In: 43rd ACM Symposium on Theory of Computing, pp. 21–30 (2011)
11. Doerr, B., Fouz, M., Friedrich, T.: Asynchronous rumor spreading in preferential attachment graphs. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 307–315. Springer, Heidelberg (2012)
12. Fountoulakis, N., Huber, A., Panagiotou, K.: Reliable broadcasting in random networks and the effect of density. In: 29th Conference on Computer Communications (IEEE INFOCOM 2010), pp. 2552–2560 (2010)
13. Fountoulakis, N., Panagiotou, K., Sauerwald, T.: Ultra-fast rumor spreading in social networks. In: Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), pp. 1642–1660 (2012)
14. Frieze, A.M., Grimmett, G.R.: The shortest-path problem for graphs with random arc-lengths. Discrete Applied Mathematics 10(1), 57–77 (1985)
15. Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: 28th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 57–68 (2011)
16. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3) (2007)
17. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In: 41st Annual Symposium on Foundations of Computer Science (FOCS 2000), pp. 565–574 (2000)
18. Klenke, A.: Probability theory: a comprehensive course. Springer (2008)
19. Massart, P.: Concentration Inequalities and Model Selection. Springer (2003)
20. Panagiotou, K., Pérez-Giménez, X., Sauerwald, T., Sun, H.: Randomized rumor spreading: the effect of the network topology (submitted)
21. Pittel, B.: On spreading a rumor. SIAM Journal on Applied Mathematics 47(1), 213–223 (1987)
22. Van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. In: Middleware, pp. 55–70 (1998)

Unit Cost Buyback Problem

Yasushi Kawase[1], Xin Han[2], and Kazuhisa Makino[3]

[1] University of Tokyo
yasushi_kawase@mist.i.u-tokyo.ac.jp
[2] Dalian University of Technology
hanxin@dlut.edu.cn
[3] Kyoto University
makino@kurims.kyoto-u.ac.jp

Abstract. In this paper, we study *unit cost buyback problem*, i.e., the buyback problem with fixed cancellation cost for each cancelled item. The input is a sequence of elements $e_1, e_2, \ldots, e_n$, each of which has a weight $w(e_i)$. We assume that weights have an upper and a lower bound, i.e., $l \leq w(e_i) \leq u$ for any i. Given the ith element e_i, we either accept e_i or reject it with no cost, subject to some constraint on the set of accepted elements. In order to accept a new element e_i, we could cancel some previous selected elements at a cost which is proportional to the number of elements canceled. Our goal is to maximize the profit, i.e., the sum of the weights of elements accepted (and not canceled) minus the total cancellation cost occurred. We construct optimal online algorithms and prove that they are the best possible, when the constraint is a matroid constraint or the unweighted knapsack constraint.

1 Introduction

In this paper, we study *unit cost buyback problem*, i.e., the buyback problem with fixed cancellation cost for each cancelled item. The buyback problem was first defined and studied in [3, 6] as a model of selling advertisement online with a buyback option. The input for the problem is a sequence of elements $e_1, e_2, \ldots, e_n$, each of which has a weight $w(e_i)$. Given the ith element e_i, we either accept e_i or reject it with no cost, subject to some constraint on the set of accepted elements. When we accept an element e_i, we could cancel some of the previously accepted elements at a cost. Our goal is to maximize the profit, i.e., the sum of the weights of elements accepted (and not canceled) minus the total cancellation cost occurred.

Examples of cancellation costs are compensatory payment, paperwork cost, and shipping charge. Compensatory payment is usually constant rate of value of cancelled items. On the other hand, paperwork cost and shipping charge usually do not depend on item value but on the number of items. In this paper, we consider the latter case.

Related Works

The buyback problem with proportional cost was studied in [1, 2, 3, 4, 5, 6, 7]. In this model, the cancellation cost of each element e_i is proportional to its weight, i.e., it is $f \cdot w(e_i)$, where $f > 0$ is a fixed constant called buyback factor. Babaioff *et al.* [3] and Constantin *et al.* [6] showed that the problem is $(1 + 2f + 2\sqrt{f(1+f)})$-competitive when the constraint is the single element constraint. Babaioff *et al.* [3] also showed that the problem has a competitive ratio $\left(1 + 2f + 2\sqrt{f(1+f)}\right)$ when the constraint is a matroid constraint.

Ashwinkumar [1] extended their results and showed that the buyback problem with the constraint of k matroid intersection is $k(1+f)(1+\sqrt{1-\frac{1}{k(1+f)}})^2$-competitive. Babaioff *et al.* [3, 4] also studied the buyback problem with the weighted knapsack constraint. They showed that if the largest element is of size at most γ, where $0 < \gamma < 1$, then the competitive ratio is $1 + 2f + 2\sqrt{f(1+f)}$ with respect to the optimum solution for the knapsack problem with capacity $(1-2\gamma)$. Han *et al.* [7] studied the buyback problem with the unweighted knapsack constraint, where knapsack problem is called unweighted if the value of each item is equal to its size. They proved this problem is $\max\{2, \frac{1+f+\sqrt{f^2+2f+5}}{2}\}$-competitive.

Unit cost buyback problem was introduced by Han *et al.* [7]. In their model, the cancellation cost of each element is a fixed constant $c > 0$. They showed tight competitive ratio for unweighted knapsack constraint case with the assumption that every element has a weight at least c, since in many applications, the cancellation cost is not higher than its value. On the other hand, in this paper, we consider a general case even when an element can have a value smaller than the cancellation cost c.

Iwama and Taketomi [11] studied the online removal knapsack problem, which can be seen as the buyback problem without cancellation cost. They obtained a $\frac{1+\sqrt{5}}{2} \approx 1.618$-competitive algorithm for the online knapsack when (i) the removable condition is allowed and (ii) the value of each element is equal to the weight, and showed that this is the best possible by providing a lower bound 1.618 for the case. We remark that the problem has unbounded competitive ratio, if at least one of the conditions (i) and (ii) is not satisfied [11, 12]. For other models of online knapsack problem such as minimum knapsack problem and knapsack problem with limited cuts, refer to papers in [8, 9, 10, 13].

Our Results

In this paper, we study the worst case analysis of buyback problem with unit cancellation cost, when the constraint is a matroid constraint, or the unweighted knapsack constraint. Let $c > 0$ be the cancellation cost of each element.

For a matroid constraint case, let $u > l > 0$ be an upper and lower bound of weight of each element. We show that the buyback problem with unit cancellation cost has the competitive ratio $\lambda(l, u, c)$ as defined below when the constraint is a matroid constraint. Let $v_\rho(n)$ satisfy a recurrence relation

$$\begin{cases} v_\rho(1) &= l, \\ v_\rho(n+1) &= \rho(v_\rho(n) - (n-1)c) \qquad (n = 1, 2, \dots) \end{cases} \tag{1}$$

and

$$v_\rho(n) = \frac{c\rho}{\rho-1}n - \left(\frac{c\rho}{(\rho-1)^2} - l\right)\rho^{n-1} - \frac{c\rho(\rho-2)}{(\rho-1)^2}.$$

Then $\lambda(l,u,c)$ is the unique value $\rho \geq 1$ which satisfies $\max_n v_\rho(n) = u$ (the uniqueness is shown in Section 3). For example, the competitive ratios $\lambda(l,u,c)$ for $(l,u) = (0.5, 1.0)$ and $(u,c) = (1.0, 0.05)$ are given in Figure 1.

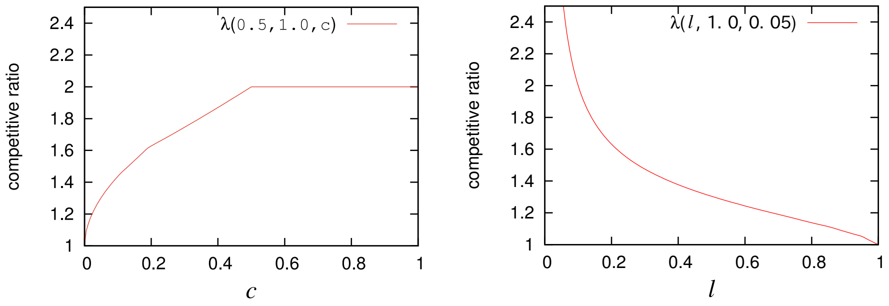


Fig. 1. The competitive ratio $\lambda(l,u,c)$ for $(l,u) = (0.5, 1.0)$ and $(u,c) = (1.0, 0.05)$

For the unweighted knapsack constraint case, let $1 > l > 0$ be a lower bound of weight of each element. This problem is a generalization of the problem dealt by Han *et al.* [7], since their problem is a special case when $l = c$. We show that the online unweighted knapsack problem with unit cancellation cost has the competitive ratio $\mu(l,c)$ in (2). Namely, we construct $\mu(l,c)$-competitive algorithms for the problem and prove that they are the best possible. Let $S_k = \{(l,c) \mid k \leq \frac{(1-l)^2}{l+c-lc} < k+1\}$ $(k = 1, 2, \dots)$ and $S_{k,1}, S_{k,2}, S_{k,3}, S_{k,4}$ $(S_k = \bigcup_{i=1}^4 S_{k,i})$ be

$$S_{k,1} = \left\{(l,c) \in S_k \mid c < \min\left\{\tfrac{2k-1}{2k(2k+1)},\, 2l - \tfrac{1}{2(k+1)}\right\}\right\},$$

$$S_{k,2} = \left\{(l,c) \in S_k \mid c \geq \tfrac{2k-1}{2k(2k+1)},\, \eta(k) > \xi(k+1),\, l+c < \tfrac{1}{k+1}\right\}$$
$$\cup \left\{(l,c) \in S_k \mid c \geq \tfrac{2k-1}{2k(2k+1)},\, \eta(k) > \tfrac{1}{(k+1)l},\, l+c \geq \tfrac{1}{k+1}\right\},$$

$$S_{k,3} = \left\{(l,c) \in S_k \mid \tfrac{1}{(k+1)l} \geq \eta(k),\, l+c \geq \tfrac{1}{k+1}\right\},$$

$$S_{k,4} = \left\{(l,c) \in S_k \mid c \geq \tfrac{2k-1}{2k(2k+1)},\, \xi(k+1) \geq \eta(k),\, l+c < \tfrac{1}{k+1}\right\}$$

(see Figures 2, 3) where

$$\eta(k) = \frac{k(c+1) + \sqrt{k^2(1-c)^2 + 4k}}{2k(1-kc)} \quad \text{and} \quad \xi(k) = \frac{kc + \sqrt{k^2c^2 + 4kl}}{2kl}.$$

Points P_k, Q_k in Figure 3 are

$$P_k = \left(\sqrt{\frac{k}{k+1}} - \frac{k}{k+1},\, 1 - \sqrt{\frac{k}{k+1}}\right),\ Q_k = \left(\frac{4k^2+2k-1}{4k(k+1)(2k+1)},\, \frac{2k-1}{2k(2k+1)}\right).$$

Let $S_{0,1} = \{(l,c) \mid \frac{(1-l)^2}{l+c-lc} < 1,\, l < \frac{1}{2},\, c < 2l - \frac{1}{2}\}$, and let $S_{0,4} = \{(l,c) \mid \frac{(1-l)^2}{l+c-lc} < 1,\, l + c < 1,\, c \geq 2l - \frac{1}{2}\}$.

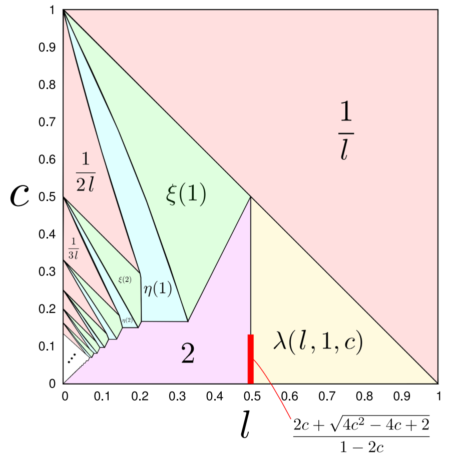


Fig. 2. The areas of the competitive ratio $\mu(l,c)$

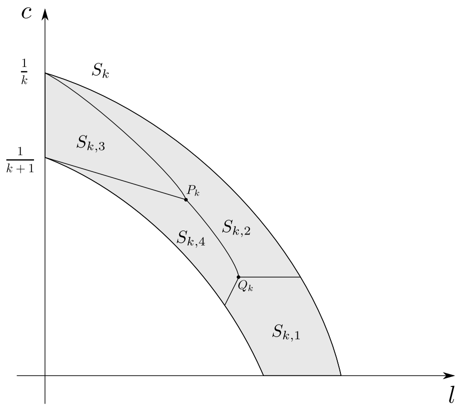


Fig. 3. The areas $S_{k,1}$, $S_{k,2}$, $S_{k,3}$, $S_{k,4}$

Then $\mu(l,c)$ is defined as

$$\mu(l,c) = \begin{cases} \frac{1}{l} & (l + c \geq 1), \\ \lambda(l,1,c) & (l > \frac{1}{2},\, l + c < 1), \\ \frac{2c+\sqrt{4c^2-4c+2}}{1-2c} & (l = 1/2,\, 1/8 > c > 0), \\ 2 & (l = \frac{1}{2},\, \frac{1}{8} \leq c < \frac{1}{2}), \\ 2 & ((l,c) \in S_{k,1},\, k = 0,1,2,\dots), \\ \eta(k) & ((l,c) \in S_{k,2},\, k = 1,2,3,\dots), \\ \frac{1}{(k+1)l} & ((l,c) \in S_{k,3},\, k = 1,2,3,\dots), \\ \xi(k+1) & ((l,c) \in S_{k,4},\, k = 0,1,2,\dots). \end{cases} \quad (2)$$

For example, the competitive ratios $\mu(l,c)$ for $l = 1/2$ and $l = c$ are given in Figure 4. The result for $l = c$ coincides with the result in [7].

The rest of the paper is organized as follows. In Section 2 we formally define unit cost buyback problem. In Section 3 we consider a matroid constraint case with an upper and lower bound of weights, and in Section 4 we consider the knapsack constraint case with lower bound of weights. Due to the space limitation, some of the proofs are omitted.

2 Preliminaries

In this section, we formally define unit cost buyback problem. Let $(E, \mathcal{I})$ be an independence system, i.e., E is a finite set and $\mathcal{I}$ is a family of subsets of E,

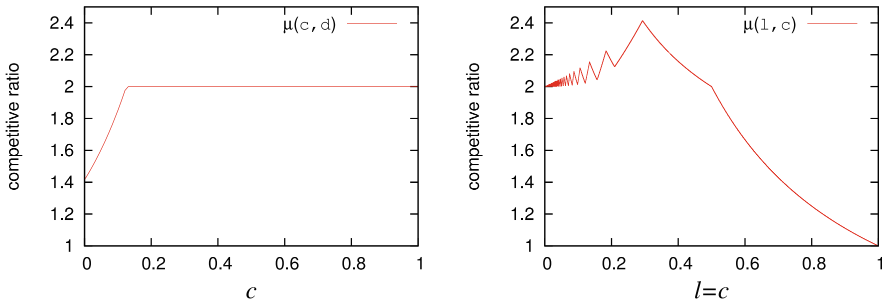


Fig. 4. The competitive ratio $\mu(l, c)$ for $l = 1/2$ and $l = c$

and $J \subseteq I \in \mathcal{I} \Rightarrow J \in \mathcal{I}$. Elements from $E = \{e_1, \ldots, e_n\}$ are presented to an algorithm in a sequential manner, and when an element e_k is presented, it must be accepted or rejected immediately. Each element e_i is associated with a weight $w(e_i)$. When e_k is accepted, the algorithm could cancel some of the previously accepted elements.

Let B_k be the set of selected elements at the end of kth round. Then $B_k \subseteq B_{k-1} \cup \{e_k\}$ and $B_k \in \mathcal{I}$. The algorithm must run based only on the weights $w(e_i)$ $(1 \leq i \leq k)$ and the feasibility of subsets $T \subseteq \{e_1, \ldots, e_k\}$. The utility of the algorithm is the total weight of the accepted elements minus the penalty paid to the canceled elements. All the canceled elements are paid a penalty c.

Let the final set held by the algorithm be $B = B_n$ and the set of elements canceled be $R = (\bigcup_i B_i) \setminus B$. Then the utility of the algorithm is defined as $\sum_{e \in B} w(e) - |R| \cdot c$.

3 Matroid Constraint Case with an Upper and Lower Bound of Weights

In this section, we consider a matroid constraint case with an upper and a lower bound of weights, where the constraint $\mathcal{I}$ is an arbitrary independence family of matroid and each element e_i has weight $l \leq w(e_i) \leq u$. We assume that $0 < l < u < \infty$.

We first show some properties about $v_\rho(n)$ in (1), and the uniqueness of $\lambda(l, u, c)$.

Proposition 1. *If* $\rho \geq 1 + \frac{c+\sqrt{c^2+4lc}}{2l}$, *then* $v_\rho(n)$ *approaches infinity as* $n \to \infty$.

Proposition 2. $\max_n v_\rho(n)$ *is continuous and strictly monotone increasing for* $1 \leq \rho < 1 + \frac{c+\sqrt{c^2+4lc}}{2l}$.

Proposition 3. *The value* $\rho \geq 1$ *which satisfies* $\max_n v_\rho(n) = u$ *is unique.*

Proof. By Propositions 1 and 2, $\max_n v_\rho(n) = u$ is unique for ρ. □

We define $v(n)$ as $v_{\lambda(l,u,c)}(n)$ and $n^* = \min\{n \mid v(n) \geq v(n+1)\}$.

Remark 1. For any positive number t, we have $\lambda(l,u,c) = \lambda(l/t, u/t, c/t)$.

Remark 2. For the case without upper bound of weights, we have $\lambda(l,\infty,c) = \lim_{u\to\infty} \lambda(l,u,c) = 1 + \frac{c+\sqrt{c^2+4lc}}{2l}$.

3.1 Upper Bound

In this subsection, we show Algorithm 1 is $\lambda(l,u,c)$-competitive for the problem. Let e_i be the element given in the ith round. Define by B_i the set of selected elements at the end of ith round, and by $w(B_i)$ the total weight in B_i.

We partition the range $[l,u]$ into the intervals

$$I_1 = [v(1), v(2)], I_2 = (v(2), v(3)], \ldots, I_{n^*-1} = (v(n^*-1), v(n^*)],$$

and let $ind(e)$ be the index of the interval e belongs to, i.e., $w(e) \in I_{ind(e)}$.

Algorithm 1. matroid constraint case

```
1: $B_0 \leftarrow \emptyset$
2: for all elements $e_i$, in order of arrival, do
3:     if $B_{i-1} \cup \{e_i\} \in \mathcal{I}$ then $B_i \leftarrow B_{i-1} \cup \{e_i\}$
4:     else let $e_i'$ be the element of smallest value such that $B_{i-1} \cup \{e_i\} \setminus \{e_i'\} \in \mathcal{I}$
5:         if $ind(e_i) > ind(e_i')$ then $B_i \leftarrow B_{i-1} \cup \{e_i\} \setminus \{e_i'\}$
6:         else $B_i \leftarrow B_{i-1}$
7: end for
```

Theorem 1. *The online Algorithm 1 is $\lambda(l,u,c)$-competitive for the unit cost buyback problem with a matroid constraint.*

Proof. Let OPT be an optimal (offline) solution for a matroid $(E,\mathcal{I})$, where $E = \{e_1,\ldots,e_n\}$. It is easy to see, OPT and B_n are bases of the matroid $(E,\mathcal{I})$. Moreover, if each element e_i has a weight $v(ind(e_i))$, B_n is a maximum-weight basis of the matroid $(E,\mathcal{I})$ with respect to weight $w' : e_i \mapsto v(ind(e_i))$ since Algorithm 1 is a matroid greedy algorithm that maximize the weight w'. Therefore, there is a perfect matching $\{(b_i^*, b_i)\}_{i=1,\ldots,h}$ such that $ind(b_i^*) = ind(b_i)$ $(i = 1,2,\ldots,h)$ for $OPT = \{b_1^*, b_2^*, \ldots, b_h^*\}$ and $B_n = \{b_1, b_2, \ldots, b_h\}$.

For each i, let k_i be $ind(b_i)$. Then, $w(b_i^*) \leq v(k_i+1)$, $w(b_i) \geq v(k_i)$, and the algorithm cancels at most $\sum_{i=1}^h (k_i - 1)$ elements. Therefore, the competitive ratio is at most

$$\frac{w(OPT)}{w(B_n) - \sum_{i=1}^h (k_i-1)c} = \frac{\sum_{i=1}^h w(b_i^*)}{\sum_{i=1}^h (w(b_i) - (k_i-1)c)} \leq \max_i \frac{w(b_i^*)}{w(b_i) - (k_i-1)c}$$
$$\leq \max_i \frac{v(k_i+1)}{v(k_i) - (k_i-1)c} = \lambda(l,u,c).$$

□

Remark 3. If l and u are not known to the algorithm in advance, we can get $\lambda(l,\infty,c)$-competitive algorithm by modifying the definition of $ind(e)$ to partition the range $[\min_{j\leq i} w(e_j), \infty]$.

3.2 Lower Bound

In this subsection, we show $\lambda(l,u,c)$ is also a lower bound for the competitive ratio of the problem for the single element case. This lower bound is applicable for the general matroid case since the single element case is a special case of it, i.e., the uniform matroid of rank 1.

Theorem 2. *There exists no online algorithm with competitive ratio less than $\lambda(l,u,c)$ for the unit cost buyback problem with the single element constraint.*

Proof. Let A denote an online algorithm chosen arbitrarily. Our adversary requests the sequence of elements whose weights are

$$v(1), v(2), \dots, v(n^*), \tag{3}$$

until A rejects some element in (3).

If A rejects the element with weight $v(1)$, then the competitive ratio of A becomes infinite. On the other hand, if A rejects the element with weight $v(k+1)$ for some $k \geq 1$, A cancels $k-1$ elements and the competitive ratio is $\frac{v(k+1)}{v(k)-(k-1)c} = \lambda(l,u,c)$. Finally, if A accepts all the elements in (3), then the competitive ratio is at least

$$\frac{v(n^*)}{v(n^*)-(n^*-1)c} = \frac{v(n^*)}{v(n^*+1)} \cdot \lambda(l,u,c) \geq \lambda(l,u,c).$$

□

4 Unweighted Knapsack Case with Lower Bound of Weights

In this section, we consider the unweighted knapsack constraint of capacity 1 case with an upper and a lower bound of weight, where the constraint $\mathcal{I} = \{I \mid \sum_{i\in I} w(e_i) \leq 1\}$ and each element e_i has weight $l \leq w(e_i) \leq 1$. We show the competitive ratio for this problem is $\mu(l,c)$ in (2). Due to the space limitation, the proof for lower bound of the competitive ratio is omitted.

Theorem 3. *There exists no online algorithm with a competitive ratio less than $\mu(l,c)$ for the unit cost buyback problem with the unweighted knapsack constraint.*

We start with several definitions and propositions needed later.

Definition 1. *We define x_k, y_k as follows:*

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

Proposition 4. *We have,*

$$\frac{1}{1-x_k-kc} = \frac{1-x_k}{kx_k} = \eta(k), \quad \frac{1}{y_k-kc} = \frac{y_k}{kl} = \xi(k).$$

Proof. We can get the results by simple calculations. □

Proposition 5. *Let* $k = \lfloor\frac{(1-l)^2}{l+c-lc}\rfloor$ *and* l, c *satisfies* $l < \frac{1}{2}$, $l + c < 1$. *Then we have* $\mu(l, c) > \frac{1}{(k+2)l}$.

Proof. If $l + c \geq \frac{1}{k+1}$, then $\mu(l, c) \geq \frac{1}{(k+1)l} \geq \frac{1}{(k+2)l}$. Otherwise, $l + c < \frac{1}{k+1}$, we have $\mu(l, c) \geq \xi(k + 1) = \frac{y_{k+1}}{(k+1)l} \geq \frac{1-l}{(k+1)l} = \max\left\{\frac{1-l}{(k+1)l}, \frac{l}{l}\right\} > \frac{1}{(k+2)l}$. □

Proposition 6. *Let* $k = \lfloor\frac{(1-l)^2}{l+c-lc}\rfloor \geq 1$. *Then we have*

$$\max\{\max_{\alpha\in\{1,2,\ldots,k\}} \eta(\alpha), 2\} = \max\{\eta(k), 2\}.$$

Proposition 7. *Let* $k = \lfloor\frac{(1-l)^2}{l+c-lc}\rfloor \geq 1$. *Then for any natural number* $\alpha \in \{1, 2, \ldots, k\}$ *and real* $x \in (0, 1 - \alpha c)$, *it holds that*

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

Proof. Since $\frac{1}{1-x-\alpha c}$ and $\frac{1-x}{\alpha x}$ are respectively monotone increasing and decreasing in x, the first inequality holds by Proposition 4. The second inequality is obtained by Proposition 6 and the definition of $\mu(l, c)$. □

Proposition 8. *Let* $k = \lfloor\frac{(1-l)^2}{l+c-lc}\rfloor \geq 1$ *and* l, c *satisfies* $l + c < \frac{1}{k+1}$. *Then for any real* $y \in ((k + 1)c, 1]$, *it holds that*

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

Proof. Since $\frac{1}{y-(k+1)c}$ and $\frac{y}{(k+1)c}$ are respectively monotone decreasing and increasing in y, the first inequality holds by Proposition 4. The second inequality follows from the definition of $\mu(l, c)$. □

4.1 Upper Bounds

In this subsection, we show that $\mu(l, c)$ is an upper bound for the competitive ratio of the problem. We consider 4 cases; the case $l + c \geq 1$ or $l = 1/2$ and $c \geq 1/8$ in Theorem 4, the case $l > 1/2$ in Theorem 5, the case $l = 1/2$ and $1/8 > c > 0$ in Theorem 6, and the remaining case $l + c < 1$ and $l < 1/2$ in Theorem 7.

Theorem 4. *There exists at most* $1/l$*-competitive algorithm for the unit cost buyback problem with the unweighted knapsack constraint.*

Proof. Consider an online algorithm which takes the first element e_1 and rejects the remaining elements. Since $w(e_1) \geq l$ and the optimal value of the offline problem is at most 1, the competitive ratio is at most $1/l$. □

Theorem 5. *There exists at most* $\lambda(l, 1, c)$*-competitive algorithm for the unit cost buyback problem with the unweighted knapsack constraint if* $l > 1/2$.

Proof. This follows from Theorem 1 since we can hold only one element in the knapsack. □

Algorithm 2. removal at most twice

```
1: $B_0 \leftarrow \emptyset$
2: for all elements $e_i$, in order of arrival, do
3:     if $w(B_{i-1}) + w(e_i) \le 1$ then $B_i \leftarrow B_{i-1} \cup \{e_i\}$ and if $w(B_i) \ge \frac{\sqrt{4c^2-4c+2}}{2}$ then
       STOP
4:     else if $w(e_i) \ge c + \frac{\sqrt{4c^2-4c+2}}{2}$ then $B_i \leftarrow \{e_i\}$ and STOP
5:     else if $w(e_i) = 1/2$ then $B_i \leftarrow \{e_i\}$
6:     else $B_i \leftarrow B_{i-1}$
7: end for
```

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

Theorem 6. *The online Algorithm 2 is* $\frac{2c+\sqrt{4c^2-4c+2}}{1-2c}$*-competitive for the unit cost buyback problem with the unweighted knapsack constraint if* $l = 1/2$ *and* $1/8 > c > 0$.

In the rest of this subsection, we would like to show the following Algorithm 3 is $\mu(l,c)$-competitive for $l + c < 1$ and $l < 1/2$. The main ideas of the algorithm are: i) it rejects elements (with no cost) many times, but in at most one round, it removes some elements from the knapsack. ii) some elements are removed from the knapsack, only when the total value in the resulting knapsack gets high enough to guarantee the optimal competitive ratio.

Algorithm 3. removal at most once

```
1: $B_0 \leftarrow \emptyset$, $l_0 \leftarrow 1/2$
2: for all elements $e_i$, in order of arrival, do
3:     if $w(e_1) \ge 1/2$ then $B_i \leftarrow \{e_1\}$ and STOP
4:     $l_i \leftarrow \min\{w(e_i), l_{i-1}\}$
5:     if $w(B_{i-1}) + w(e_i) \le 1$ then $B_i \leftarrow B_{i-1} \cup \{e_i\}$ and if $w(B_i) \ge 1/\mu(l_i,c)$ then
       STOP
6:     else if $\exists B'_{i-1} \subseteq B_{i-1}$ s.t. $\frac{1}{\mu(l_i,c)} + |B_{i-1} \setminus B'_{i-1}|c \le w(B'_{i-1}) + w(e_i) \le 1$ then
       $B_i \leftarrow B'_{i-1} \cup \{e_i\}$ and STOP
7:     else $B_i \leftarrow B_{i-1}$
8: end for
```

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

Lemma 1. *If* $w(B_{i-1}) + w(e_i) > 1$ *and some* $B'_{i-1} \subseteq B_{i-1}$ *satisfies* $\mu(l_i,c) \cdot w(B_{i-1}) < w(B'_{i-1}) + w(e_i) \le 1$*, then the sixth line is executed in the ith round.*

Proof. We assume that B'_{i-1} is maximal for $w(B'_{i-1}) + w(e_i) \le 1$. Let $k = \lfloor \frac{(1-l_i)^2}{l_i+c-l_ic} \rfloor$ and

$$\alpha = |B_{i-1} \setminus B'_{i-1}|, \quad 1 - x = y = w(B'_{i-1}) + w(e_i).$$

As $\mu(l_i,c) \cdot w(B_{i-1}) < w(B'_{i-1}) + w(e_i) \le 1$, we have

$$\mu(l_i,c) < \frac{w(B'_{i-1}) + w(e_i)}{w(B_{i-1})} = \frac{w(B'_{i-1}) + w(e_i)}{w(B'_{i-1}) + w(B_{i-1} \setminus B'_{i-1})} \le \frac{w(B'_{i-1}) + w(e_i)}{w(B_{i-1} \setminus B'_{i-1})}.$$

Since $w(B_{i-1} \setminus B'_{i-1}) \geq \alpha l_i$ and $w(B_{i-1} \setminus B'_{i-1}) \geq \alpha x$ as B'_{i-1} is maximal, we have $\mu(l_i, c) < \frac{1-x}{\alpha x}$ and $\mu(l_i, c) < \frac{y}{\alpha l_i}$.

For the cardinality of B_{i-1}, we have $|B_{i-1}| \leq k+1$ since $|B_{i-1}| \geq k+2$ implies $\frac{1}{w(B_{i-1})} \leq \frac{1}{(k+2)l_i} \leq \mu(l_i, c)$ by Proposition 5.

If $\alpha \leq k$, then we have $\frac{1}{w(B'_{i-1})+w(e_i)-\alpha \cdot c} = \frac{1}{1-x-\alpha \cdot c} \leq \eta(\alpha) \leq \mu(l_i, c)$ by Proposition 7 and $\mu(l_i, c) < \frac{1-x}{\alpha x}$. On the other hand, if $\alpha = k+1$, then we have $|B_{i-1}| = k+1$ and we can assume that $l_i + c < \frac{1}{k+1}$ since $l_i + c \geq \frac{1}{k+1}$ implies $\frac{1}{w(B_{i-1})} \leq \frac{1}{(k+1)l_i} \leq \mu(l_i, c)$ which contradicts the assumption. Therefore, we obtain $\frac{1}{w(B'_{i-1})+w(e_i)-\alpha \cdot c} = \frac{1}{y-\alpha \cdot c} \leq \xi(k+1) \leq \mu(l_i, c)$ by Proposition 8 and $\mu(l_i, c) < \frac{y}{\alpha l_i}$. □

Let OPT_i denote an optimal solution whose input sequence is $e_1, \ldots, e_i$.

Lemma 2. *If $w(B_i) < 1/\mu(l_i, c)$ then we have $|OPT_i \setminus B_i| \leq 1$.*

Proof. B_i contains all the elements smaller than $1/2$, since $w(B_i) < 1/\mu(l_i, c) \leq 1/2$. Any element $e \in OPT_i \setminus B_i$ has weight greater than $1 - 1/\mu(l_i, c) \geq 1/2$. Therefore, $|OPT_i \setminus B_i| \leq 1$ holds by $w(OPT_i) \leq 1$. □

Theorem 7. *The online Algorithm 3 is $\mu(l, c)$-competitive for the unit cost buy-back problem with the unweighted knapsack constraint if $l < 1/2$.*

Proof. If $w(e_1) \geq 1/2$, then $w(B_i) = w(e_1) \geq 1/2$, and the competitive ratio is at most $1/w(B_i) \leq 2 \leq \mu(l, c)$. Thus, we assume that $w(e_1) < 1/2$.

Suppose that the sixth line is executed in round k. Then it holds that $\frac{1}{\mu(l_k, c)} + |B_{k-1} \setminus B'_{k-1}|c \leq w(B'_{k-1}) + w(e_k) = w(B_k)$. Since $w(B_i) = w(B_k)$ holds for all $i \geq k$, we have

$$\frac{w(OPT_i)}{w(B_i) - |B_{k-1} \setminus B'_{k-1}|c} \leq \frac{1}{w(B'_{k-1}) + w(e_k) - |B_{k-1} \setminus B'_{k-1}|c} < \mu(l_k, c) \leq \mu(l, c).$$

We next assume that the sixth line has never been executed. If $w(B_i) \geq 1/\mu(l_i, c)$, we have the competitive ratio $w(OPT_i)/w(B_i) \leq 1/w(B_i) \leq \mu(l_i, c) \leq \mu(l, c)$. On the other hand, if $w(B_i) < 1/\mu(l_i, c)$, $|OPT \setminus B_i| = 0$ or 1 holds by Lemma 2. If $|OPT_i \setminus B_i| = 0$, we obtain the competitive ratio 1. Otherwise, i.e., $OPT_i \setminus B_i = \{e_k\}$ for some k, Lemma 1 implies that $\mu(l_k, c) \cdot w(B_{k-1}) \geq w(B'_{k-1}) + w(e_k)$ for $B'_{k-1} = OPT_i \cap B_{k-1}$. Therefore we obtain,

$$\begin{aligned}
\frac{w(OPT_i)}{w(B_i)} &\leq \frac{w(B_{k-1} \cap OPT_i) + w(e_k) + w(B_i \setminus B_{k-1})}{w(B_{k-1}) + w(B_i \setminus B_{k-1})} \\
&\leq \max\left\{\frac{w(B_{k-1} \cap OPT_i) + w(e_k)}{w(B_{k-1})}, 1\right\} \\
&= \frac{w(B'_{k-1}) + w(e_k)}{w(B_{k-1})} \leq \mu(l_k, c) \leq \mu(l, c).
\end{aligned}$$

□

Remark 4. We can almost always get $\mu(l, c)$-competitive algorithm even if l is not known in advance. If $w(e_1) \geq 1/2$, Algorithm 1 for $l = w(e_1)$ and $u = 1$ is $\lambda(l, 1, c)$-competitive for $l > 1/2$ and 2-competitive for $l \leq 1/2$. Note that, $\mu(l, c) = \lambda(l, 1, c)$ for $l > 1/2$ and $\mu(l, c) \geq 2$ for $l < 1/2$. If $w(e_1) < 1/2$, Algorithm 3 is $\mu(l, c)$-competitive.

Acknowledgment. This work is partially supported by Grant-in-Aid for Scientific research from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and JST, ERATO, Kawarabayashi Large Graph Project, and NSFC(11101065) and "the Fundamental Research Funds for the Central Universities" (DUT12LK09) in China.

References

1. Badanidiyuru Varadaraja, A.: Buyback problem - approximate matroid intersection with cancellation costs. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 379–390. Springer, Heidelberg (2011)
2. Ashwinkumar, B.V., Kleinberg, R.: Randomized online algorithms for the buyback problem. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 529–536. Springer, Heidelberg (2009)
3. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling banner ads: Online algorithms with buyback. In: Proceedings of 4th Workshop on Ad Auctions (2008)
4. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling ad campaigns: Online algorithms with cancellations. In: ACM Conference on Electronic Commerce, pp. 61–70 (2009)
5. Biyalogorsky, E., Carmon, Z., Fruchter, G.E., Gerstner, E.: Research note: Overselling with opportunistic cancellations. Marketing Science 18(4), 605–610 (1999)
6. Constantin, F., Feldman, J., Muthukrishnan, S., Pál, M.: An online mechanism for ad slot reservations with cancellations. In: SODA, pp. 1265–1274 (2009)
7. Han, X., Kawase, Y., Makino, K.: Online knapsack problem with removal cost. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 61–73. Springer, Heidelberg (2012)
8. Han, X., Kawase, Y., Makino, K.: Randomized algorithms for removable online knapsack problems. In: Fellows, M., Tan, X., Zhu, B. (eds.) FAW-AAIM 2013. LNCS, vol. 7924, pp. 60–71. Springer, Heidelberg (2013)
9. Han, X., Makino, K.: Online minimization knapsack problem. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 182–193. Springer, Heidelberg (2010)
10. Han, X., Makino, K.: Online removable knapsack with limited cuts. Theoretical Computer Science 411, 3956–3964 (2010)
11. Iwama, K., Taketomi, S.: Removable online knapsack problems. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 293–305. Springer, Heidelberg (2002)
12. Iwama, K., Zhang, G.: Optimal resource augmentations for online knapsack. In: APPROX-RANDOM, pp. 180–188 (2007)
13. Noga, J., Sarbua, V.: An online partially fractional knapsack problem. In: ISPAN, pp. 108–112 (2005)

Faster Rumor Spreading with Multiple Calls

Konstantinos Panagiotou[1], Ali Pourmiri[2], and Thomas Sauerwald[2,3]

[1] The Ludwig Maximilian University of Munich, Germany
[2] Max Planck Institute for Informatics, Saarbrücken, Germany
[3] University of Cambridge, United Kingdom

Abstract. We consider the random phone call model introduced by Demers et al. [8], which is a well-studied model for information dissemination on networks. One basic protocol in this model is the so-called Push protocol which proceeds in synchronous rounds. Starting with a single node which knows of a rumor, every informed node calls a random neighbor and informs it of the rumor in each round. The Push-Pull protocol works similarly, but additionally every uninformed node calls a random neighbor and may learn the rumor from that neighbor.

While it is well-known that both protocols need $\Theta(\log n)$ rounds to spread a rumor on a complete network with n nodes, we are interested by how much we can speed up the spread of the rumor by enabling nodes to make more than one call in each round. We propose a new model where the number of calls of a node u is chosen independently according to a probability distribution R with bounded mean determined at the beginning of the process. We provide both lower and upper bounds on the rumor spreading time depending on statistical properties of R such as the mean or the variance. If R follows a power law distribution with exponent $\in (2, 3)$, we show that the Push-Pull protocol spreads a rumor in $\Theta(\log \log n)$ rounds.

1 Introduction

Rumor spreading is an important primitive for information dissemination in networks. The goal is to spread a piece of information, the so-called rumor, from an arbitrary node to all the other nodes. The random phone call model is based on the simple idea that every node picks a random neighbor and these two nodes are able to exchange information in that round. This paradigm ensures that the protocol is local, scalable and robust against network failures (cf. [11]). Therefore these protocols have been successfully applied in other contexts such as replicated databases [8], failure detection [23], resource discovery [17], load balancing [3], data aggregation [19], and analysis of the spread of computer viruses [2].

A basic protocol for spreading a rumor in the phone call model is the Push protocol. At the beginning, there is a single node who knows of some rumor. Then in each of the following rounds every *informed* node calls a random neighbor chosen independently and uniformly at random and informs it of the rumor. The Pull protocol is symmetric, here every *uninformed* node calls a random neighbor chosen independently and uniformly at random, and if that neighbor happens to be informed the node becomes informed. The Push-Pull protocol is simply the combination of both protocols. Most studies in randomized rumor spreading concern the runtime of these protocols which is the number of rounds required until a rumor initiated by a single node reaches all other nodes.

In one of the first papers in this area, Frieze and Grimmett [14] proved that if the underlying graph is a complete graph with n nodes, then the runtime of the Push protocol is $\log_2 n + \log n \pm o(\log n)$ with high probability[1], where log denotes the natural logarithm. This result was later strengthened by Pittel [22]. For the standard Push-Pull protocol, Karp et al. [18] proved a runtime bound of $\log_3 n + \mathcal{O}(\log\log n)$. In order to overcome the large number of $\Theta(n \log n)$ calls, Karp et al. also presented an extension of the Push-Pull protocol together with a termination mechanism that spreads a rumor in $\mathcal{O}(\log n)$ rounds using only $\mathcal{O}(n \log\log n)$ messages. More recently Doerr and Fouz [9] proposed a new protocol using only Push calls with runtime $(1+o(1))\log_2 n$ using only $O(n \cdot f(n))$ calls (and messages), where $f(n)$ is an arbitrarily slow growing function.

Besides the complete graph, the randomized rumor spreading protocols mentioned above have been shown to be efficient also on other topologies. In particular, their runtime is at most logarithmic in the number of nodes n for topologies ranging from basic networks, such as random graphs [11,12] and hypercubes [11], random regular graphs [1], graphs with constant conductance [20,5,15], constant weak conductance [4] or constant vertex expansion [16], to more complex structures including preferential attachment graphs modeling social networks [10,13]. In particular, recent studies establishing a sub-logarithmic runtime on certain social network models [10,13] raise the question whether it is possible to achieve a sub-logarithmic runtime also on complete graphs. In addition to analyses on static graphs, there are also studies on mobile geometric graphs, e.g., [7,21], that have deal with strong correlations as nodes are moving according to a random walk.

Since the Push protocol, the Pull protocol and the Push-Pull protocol all require $\Theta(\log n)$ rounds to spread the rumor on a complete graph, we equip nodes with the possibility of calling more than one node in each round. Specifically, we assume that the power of each node u, denoted by C_u is determined by a probability distribution R on the positive integers which is independent of u. In order to keep the overall communication cost small, we focus on distributions R satisfying $\sum_{u \in V} C_u = \mathcal{O}(n)$ with high probability – in particular, R has bounded mean. While being a natural extension from a theoretical perspective, different C_u values could arise due to varying battery capacities, processor speeds or clock synchronizations. Our aim is to understand the impact of the distribution R on the runtime of randomized rumor spreading. In particular, we seek for conditions on R which are necessary (or sufficient) for a sublogarithmic runtime.

Our first result concerns the Push protocol for the case where R has bounded mean and bounded variance. As this is the most basic setting, our runtime bound

[1] By with high probability we refer to an event which holds with probability $1 - o(1)$ as $n \to \infty$. For simplicity, we sometimes omit the "with high probability" in the introduction.

is even tight up to low-order terms. To this end, let $T_n = \min\{t \mid \mathbf{Pr}[I_t = n] \geqslant 1 - g(n)\}$ be the first round in which all nodes are informed with probability $1 - g(n)$, where $g(n)$ is a function tending to zero as n goes to infinity (for simplicity we do not specify $g(n)$).

Theorem 1.1. *Consider the* Push *protocol and let* R *be a distribution with* $\mathbf{E}[R] = \mathcal{O}(1)$ *and* $\mathbf{Var}[R] = \mathcal{O}(1)$. *Then* $|T_n - (\log_{1+\mathbf{E}[R]} n + \log_{e^{\mathbf{E}[R]}} n)| = o(\log n)$.

Note that by putting $R \equiv 1$, we retain the classic result by Frieze and Grimmett for the standard Push protocol. If we drop the assumption on the variance, then the theorem below provides a lower bound of $\Omega(\log n)$. Although this result is less precise than Theorem 1.1, it demonstrates that it is necessary to consider the Push-Pull protocol in order to achieve a sub-logarithmic runtime.

Theorem 1.2. *Assume that* R *is any distribution with* $\mathbf{E}[R] = \mathcal{O}(1)$. *Then with prob.* $1 - o(1)$, *the* Push *protocol needs* $\Omega(\log n)$ *rounds to inform all nodes.*

We point out that the lower bound in Theorem 1.2 is tight up to constant factors, as the results in [14,22] for the standard Push-Pull protocol already imply an upper bound of $\mathcal{O}(\log n)$ rounds. Next we consider the Push-Pull protocol and extend the lower bound of $\Omega(\log n)$ from Theorem 1.1.

Theorem 1.3. *Assume that* R *is any distribution with* $\mathbf{E}[R] = \mathcal{O}(1)$ *and* $\mathbf{Var}[R] = \mathcal{O}(1)$. *Then for any constant* $\epsilon > 0$, *with probability* $1-\epsilon$ *the* Push-Pull *protocol needs at least* $\Omega(\log n)$ *rounds to inform all nodes.*

Theorem 1.3 establishes that an unbounded variance is necessary to break the $\Omega(\log n)$ lower bound. An important distribution with bounded mean but unbounded variance is the *power law distribution* with exponent $\beta < 3$, i.e., there are constants $0 < c_1 \leqslant c_2$ such that $c_1 z^{1-\beta} \leqslant \mathbf{Pr}[R \geqslant z] \leqslant c_2 z^{1-\beta}$ for any $z \geqslant 1$, and $\mathbf{Pr}[R \geqslant 1] = 1$. We are especially interested in power law distributions, because they are scale invariant and have been observed in a variety of settings in real life. Our main result below shows that this natural distribution achieves a sublogarithmic runtime.

Theorem 1.4. *Assume that* R *is a power law distribution with* $2 < \beta < 3$. *Then the* Push-Pull *protocol informs all nodes in* $\Theta(\log\log n)$ *rounds with prob.* $1-o(1)$.

Notice that if R is a power law distribution with $\beta > 3$, then Theorem 1.3 applies because the variance of R is bounded. Hence our results reveal a dichotomy in terms of the exponent β: if $2 < \beta < 3$, then the Push-Pull protocol finishes in $\mathcal{O}(\log\log n)$ rounds, whereas for $\beta > 3$ the Push-Pull protocol finishes in $\Theta(\log n)$ rounds [2]. While a very similar dichotomy was shown in [13] for random graphs with a power law degree distribution, our result here concerns the spread of the rumor from one to *all* nodes (and not only to a constant fraction as in [13]).

[2] We do not consider the case $\beta \leqslant 2$, since then there exists at least one node with degree $\Omega(n)$ and the rumor is spread in constant time (additionally, $\mathbf{E}[R]$ is no longer bounded). The analysis of the case $\beta = 3$ is an interesting open problem.

In addition, the distribution of the edges used throughout the execution of the Push-Pull protocol is different from the distribution of the edges in a power law random graph, as the latter is proportional to the product of the two nodes weights. Therefore it seems difficult to apply the previous techniques for power law random graphs used for the average distance [6] and rumor spreading [13].

Besides the power law distribution, one may also consider a simple two point distribution, where for instance, $R = n$ with probability n^{-1} and $R = 1$ otherwise. It is then straightforward to see that with constant probability, the Push-Pull protocol informs all nodes in $\mathcal{O}(1)$ rounds. The same result also holds if $R = n^{\epsilon}$ with probability $n^{-\epsilon}$ and $R = 1$ otherwise. However, the power law distribution is arguably a more natural distribution which occurs in a variety of instances in practice.

Finally, we also show that it is crucial that the C_u's do not change over time. Instead, suppose we generate a new variable C_u^t according to the distribution R for the number of calls made by node u in each round t. Then one can prove a lower bound of $\Omega(\log n)$ for the Push-Pull protocol for any distribution R with bounded mean. Based on this lower bound it seems crucial to have a *fixed* set of powerful nodes (i.e. nodes u with large C_u) in order to obtain a sublogarithmic rumor spreading time.

2 Definitions and Notations

We now provide additional definitions and notations (note that the classic Push, Pull and Push-Pull protocols have already been defined before). Here we generalize the classic Push, Pull and Push-Pull to the following statistical model on a complete graph with n nodes.

Before the protocol starts, every node u generates a random integer $C_u \geqslant 1$ according to a distribution R. Then, the rumor is placed on a randomly chosen node[3]. Our generalized Push, Pull and Push-Pull protocol proceed like the classic ones except that every (un)informed node u calls C_u node(s) chosen independently and uniformly at random and sends (request) the rumor.

Let $\mathcal{I}_t$ be the set of all informed nodes in round t (which means after the execution of round t) and $\mathcal{U}_t$ be the complement of $\mathcal{I}_t$, i.e., the set of uninformed nodes. The size of $\mathcal{I}_t$ and $\mathcal{U}_t$ are denoted by I_t and U_t. We indicate the set of newly informed nodes in round $t+1$ by $\mathcal{N}_t$ and its size is denoted by N_t. Let S_t be the number of Push calls in round $t+1$, so $S_t = \sum_{u \in \mathcal{I}_t} C_u \geqslant I_t$. Let us define $\mathcal{N}_t^{\mathsf{Pull}}$ and $\mathcal{N}_t^{\mathsf{Push}}$ to be the set of newly informed nodes by Pull and Push calls in round $t+1$, respectively. The size of $\mathcal{N}_t^{\mathsf{Pull}}$ and $\mathcal{N}_t^{\mathsf{Push}}$ are denoted by N_t^{Pull} and N_t^{Push}. The size of every set divided by n will be denoted by the

[3] This is equivalent to saying that the initial node which knows the rumor has to be chosen without knowing the sequence $C_u, u \in \mathcal{V}$. We make this assumption throughout the paper, as it is frequently needed for lower bounding the runtime, e.g., the lower bound in Theorem 1.2 may not hold if the rumor initiates from the node with the largest C_u.

corresponding small letter, so i_t, n_t and s_t are used to denote I_t/n, N_t/n, and S_t/n, respectively. Further, we define the set

$$\mathcal{L}(z) := \{u \in \mathcal{V} : C_u \geqslant z\}.$$

The size of $\mathcal{L}(z)$ is denoted by $L(z)$. We define Δ to be $\max_{u \in \mathcal{V}} C_u$.

3 Push Protocol

3.1 Push Protocol with Bounded Variance (Thm. 1.1)

In this subsection we assume that the random numbers C_u's are generated according to some distribution R with bounded mean and variance. Recall that $T_n := \min\{t \mid \mathbf{Pr}[I_t = n] \geqslant 1 - o(1)\}$, i.e., the first round in which all nodes are informed with probability $1 - o(1)$. In Theorem 1.1 we show that if R is a distribution with $\mathbf{E}[R] = \mathcal{O}(1)$ and $\mathbf{Var}[R] = \mathcal{O}(1)$, then $|T_n - (\log_{1+\mathbf{E}[R]} n + \log_{\mathrm{e}^{\mathbf{E}[R]}} n)| = o(\log n)$.

To prove this result, we study the protocol in three consecutive phases. In the following we give a brief overview of the proof.

- **The Preliminary Phase.** This phase starts with one informed node and ends when $I_t \geqslant \log^5 n$ and $S_t \leqslant \log^{\mathcal{O}(1)} n$. Similar to the Birthday Paradox we show that in each round every Push call informs a different uninformed node and thus the number of informed nodes increases by $S_t \geqslant I_t$. Hence after $\mathcal{O}(\log\log n)$ rounds there are at least $\log^5 n$ informed nodes. Further, since $\mathbf{E}[R] = \mathcal{O}(1)$, after $\mathcal{O}(\log\log n)$ rounds we also have $S_t \leqslant \log^{\mathcal{O}(1)} n$.
- **The Middle Phase.** This phase starts when $\log^5 n \leqslant I_t \leqslant S_t \leqslant \log^{\mathcal{O}(1)} n$ and ends when $I_t \geqslant \frac{n}{\log\log n}$. First we show that the number of Push calls S_t increases by a factor of approximately $1 + \mathbf{E}[R]$ as long as the number of informed nodes is $o(n)$. Then we prove that the number of newly informed nodes in round $t + 1$ is roughly the same as S_t. Therefore an inductive argument shows that it takes $\log_{1+\mathbf{E}[R]} n \pm o(\log n)$ rounds to reach $\frac{n}{\log\log n}$ informed nodes.
- **The Final Phase.** This phase starts when $I_t \geqslant \frac{n}{\log\log n}$ and ends when all nodes are informed with high probability. In this phase, we first prove that after $o(\log n)$ rounds the number of uninformed nodes decreases to $\frac{n}{\log^5 n}$. Then we show the probability that an arbitrary uninformed node remains uninformed is $e^{-\mathbf{E}[R] \pm o(\frac{1}{\log n})}$, so U_t decreases by this probability. Finally, an inductive argument establishes that it takes $\log_{\mathrm{e}^{\mathbf{E}[R]}} n \pm o(\log n)$ rounds until every node is informed.

3.2 Push Protocol with Arbitrary Variance (Thm. 1.2)

We prove that if R is any distribution with $\mathbf{E}[R] = \mathcal{O}(1)$, then with probability $1 - o(1)$ the Push protocol needs at least $\Omega(\log n)$ rounds to inform all nodes. In the Push protocol, in round $t + 1$, at most S_t randomly chosen uninformed nodes are informed. Hence the total contribution of newly informed nodes to $\mathbf{E}[S_{t+1}]$ is at most $\mathbf{E}[R] \cdot S_t$. Applying the law of total expectation shows that $\mathbf{E}[S_{t+1}] \leqslant (1+\mathbf{E}[R])^t \, \mathbf{E}[R]$ which implies that $\Omega(\log n)$ rounds are necessary to inform all nodes.

4 Push-Pull Protocol

4.1 Push-Pull Protocol with Bounded Variance (Thm 1.3)

In this part we consider the case where R is a distribution with bounded mean and bounded variance. We prove that with probability at least $1-\epsilon$, the Push-Pull protocol needs at least $\Omega(\log n)$ rounds to inform all nodes. One interesting example for a distribution R with bounded mean and bounded variance is a power law distribution with parameter $\beta > 3$.

The crucial ingredient of the proof is to bound the C_u's of the nodes that become informed by using Pull, i.e., the C_u's of uninformed nodes that call an informed node. Note that the contribution of an uninformed node $u \in \mathcal{U}_t$ to $\mathbf{E}[S_{t+1}]$ is C_u times the probability that it gets informed, which is at most $C_u \cdot (I_t/n) \leqslant C_u \cdot (S_t/n)$. Hence the contribution of $u \in \mathcal{U}_t$ is at most $C_u^2 \cdot (S_t/n)$. Now using the assumption that R has bounded variance, we have that $\sum_{u \in \mathcal{V}} C_u^2 = \mathcal{O}(n)$ which implies that S_t increases only exponentially in t.

4.2 Push-Pull Protocol with Power Law Distr. $2 < \beta < 3$ (Thm. 1.4)

In this section we analyze the Push-Pull protocol where R is a power law distribution with $2 < \beta < 3$ and show that it only takes $\Theta(\log\log n)$ rounds to inform all with probability $1-o(1)$.

To prove the upper bound of $\mathcal{O}(\log\log n)$, we study the protocol in three consecutive phases and show each phase takes only $\mathcal{O}(\log\log n)$ rounds. The proof of the lower bound is ommitted in this extended abstract.

Proof of the Upper Bound. The following lemmas about Push will be used throughout this section.

Lemma 4.1. *Consider the* Push *protocol and suppose that $S_t \leqslant \log^c n$, where $c > 0$ is any constant. Then with probability $1-\mathcal{O}(\frac{\log^{2c} n}{n})$ we have $I_{t+1} = I_t + S_t$.*

Lemma 4.2. *Consider the* Push *protocol.Then with probability $1-o(\frac{1}{\log n})$ we have that $s_t - 2s_t^2 - 2\sqrt{\frac{s_t \log\log n}{n}} \leqslant n_t \leqslant s_t$.*

We will also use the following fact about Power law distributions.

Lemma 4.3. *Let $\{C_u \colon u \in \mathcal{V}\}$ be a set of n independent random variables and assume that each C_u is generated according to a power law distribution with exponent $\beta > 2$. Then for every $z = \mathcal{O}(n^{\frac{1}{\beta-1}}/\log n)$, it holds with probability $1-o(\frac{1}{n})$*

$$\frac{n \cdot c_1 \cdot z^{1-\beta}}{2} \leqslant L(z) \leqslant \frac{3 \cdot n \cdot c_2 \cdot z^{1-\beta}}{2}.$$

The Preliminary Phase. This phase starts with just one informed node and ends when $I_t \geqslant n^{\frac{1}{\beta-1}}/(2\log n)$. Let T_1 be the number of rounds needed so that the number of informed nodes exceeds $n^{\frac{1}{\beta-1}}/(2\log n)$. We will show that with probability $1-o(1)$, $T_1 = \mathcal{O}(\log\log n)$. At first we prove the following lemma.

Lemma 4.4. *Let* $c > 0$ *be any constant. Then with probability* $1-o(1)$*, the number of rounds needed to inform* $\log^c n$ *nodes is bounded by* $\mathcal{O}(\log\log n)$*.*

Proof. In order to prove our lemma we only consider Push calls and apply Lemma 4.1 which states that as long as $S_t \leqslant \log^c n$, with probability $1-\mathcal{O}(\frac{\log^{2c} n}{n})$,

$$I_{t+1} = I_t + S_t \geqslant 2I_t.$$

Thus as long as $S_t \leqslant \log^c n$, in each round the number of informed nodes is at least doubled. So we conclude that with probability $1-o(1)$, $\mathcal{O}(\log\log n)$ rounds are sufficient to inform $\log^c n$ nodes. $\square$

Lemma 4.5. *With probability* $1-o(1)$*,* $T_1 = \mathcal{O}(\log\log n)$*.*

Proof. Let T_0 be the first round when $I_{T_0} \geqslant \log^{\frac{2}{3-\beta}} n$. Let us define the constant $\gamma := \frac{3-\beta}{2(\beta-2)} > 0$. Let T be the first round such that

$$I_{T-1}^{(1+\gamma)} \leqslant n^{\frac{1}{\beta-1}}/\log n < I_T^{(1+\gamma)}.$$

Now for any $T_0 \leqslant t \leqslant T$, we can apply Lemma 4.3 and conclude that with probability $1-o(\frac{1}{n})$,

$$\sum_{u\in\mathcal{L}(I_t^{1+\gamma})} C_u \geqslant L(I_t^{1+\gamma}) \cdot I_t^{1+\gamma} \geqslant \frac{n\cdot c_1 \cdot I_t^{(1+\gamma)(2-\beta)}}{2}.$$

So,

$$\frac{I_t}{n}\sum_{u\in\mathcal{L}(I_t^{1+\gamma})} C_u \geqslant \frac{c_1\cdot I_t^{1+(1+\gamma)(2-\beta)}}{2} = \frac{c_1\cdot I_t^{3-\beta+\gamma(2-\beta)}}{2}.$$

We will bound the probability that none of $u \in \mathcal{L}(I_t^{1+\gamma})$ gets informed by Pull calls in round $t+1$ as follows,

$$\prod_{u\in\mathcal{L}(I_t^{1+\gamma})}\left(1-\frac{I_t}{n}\right)^{C_u} = \left(1-\frac{I_t}{n}\right)^{\sum_{u\in\mathcal{L}(I_t^{1+\gamma})} C_u} \leqslant e^{-c_1\cdot I_t^{3-\beta+\gamma(2-\beta)}} = e^{-c_1\cdot I_t^{\frac{3-\beta}{2}}}.$$

Since for any $t \geqslant T_0$, $I_t \geqslant \log^{\frac{2}{3-\beta}} n$, we have that with probability at least $1-n^{-c_1}$, at least one node in $\mathcal{L}(I_t^{1+\gamma})$ gets informed by Pull in round $t+1$. Hence we have that $S_{t+1} \geqslant I_t^{1+\gamma}$. Let us now consider the Push calls in round $t+2$.

By applying Lemma 4.1 we know that as long as $S_{t+1} = o(n)$ with probability $1 - o(\frac{1}{\log n})$, $S_{t+1}(1 - o(1)) \leqslant N_{t+1}$. Thus,

$$I_{t+2} \geqslant I_{t+1} + S_{t+1}(1 - o(1)) > \frac{I_t^{1+\gamma}}{2}.$$

An inductive argument shows that for any integer $k \geqslant 1$ as long as $I_{T_0+2k-2}^{1+\gamma} \leqslant n^{\frac{1}{\beta-1}} / \log n$, with probability $1 - o(\frac{k}{\log n})$

$$I_{T_0+2k} > \left(\frac{1}{2}\right)^{\sum_{i=0}^{k-1}(1+\gamma)^i} I_{T_0}^{(1+\gamma)^k} = \left(\frac{I_{T_0}}{2^\gamma}\right)^{(1+\gamma)^k} \cdot 2^{1/\gamma} > \left(\frac{\log^{\frac{2}{3-\beta}} n}{C'}\right)^{(1+\gamma)^k},$$

where $C' = 2^\gamma = \mathcal{O}(1)$. So we conclude that after $T_0 + 2k$ rounds, where $k = o(\log_{1+\gamma} \log n)$, there are two cases: either $I_{T_0+2k} \geqslant n^{\frac{1}{\beta-1}}/(2\log n)$ which means $T_1 \leqslant T_0 + 2k = \mathcal{O}(\log\log n)$ and we are done, or

$$I_{T_0+2k} < n^{\frac{1}{\beta-1}}/(2\log n) < n^{\frac{1}{\beta-1}} / \log n < I_{T_0+2k}^{1+\gamma}.$$

In the latter case, we change the value γ to γ' which satisfies $I_{T_0+2k}^{1+\gamma'} = n^{\frac{1}{\beta-1}} / \log n$ and a similar argument shows that

$$I_{T_0+2k+2} \geqslant n^{\frac{1}{\beta-1}}/(2\log n).$$

□

The Middle Phase. This phase starts with at least $n^{\frac{1}{\beta-1}}/(2\log n)$ informed nodes and ends when $I_t \geqslant \frac{n}{\log n}$. Let T_2 be the first round in which $\frac{n}{\log n}$ nodes are informed. We will show that $T_2 - T_1 = \mathcal{O}(\log\log n)$. In contrast to the Preliminary Phase where we focus only on an informed node with maximal C_u, we now consider the number of informed nodes u with a C_u above a certain threshold Z_{t+1} which is inversely proportional to I_t.

Lemma 4.6. *Suppose that $I_t \geqslant n^{\frac{1}{\beta-1}}/(2\log n)$ for some round t. Let $Z_{t+1} := \frac{n \log\log n}{I_t}$. Then with probability $1 - o(\frac{1}{n})$,*

$$|\mathcal{L}(Z_{t+1}) \cap \mathcal{I}_{t+1}| \geqslant \frac{1}{4} L(Z_{t+1}).$$

Proof. We consider two cases. If at least $\frac{1}{4}$ of the nodes in $\mathcal{L}(Z_{t+1})$ are already informed (before round $t+1$), then the statement of the lemma is true. Otherwise $|\mathcal{L}(Z_{t+1}) \cap \mathcal{U}_{t+1}| > \frac{3}{4} L(Z_{t+1})$. In the latter case, we define

$$\mathcal{L}'(Z_{t+1}) := \mathcal{L}(Z_{t+1}) \cap \mathcal{U}_{t+1}.$$

Let X_u be an indicator random variable for every $u \in \mathcal{L}'(Z_{t+1})$ so that $X_u = 1$ if u gets informed by Pull in round $t + 1$ and $X_u = 0$ otherwise.

Then we define a random variable X to be $X := \sum_{u\in\mathcal{L}'(Z_{t+1})} X_u$. Since for every $u \in \mathcal{L}'(Z_{t+1})$, $C_u \geqslant Z_{t+1} = \frac{n\log\log n}{I_t}$, it follows that

$$\mathbf{Pr}\left[X_u = 1\right] = 1 - \left(1 - \frac{I_t}{n}\right)^{C_u} \geqslant 1 - \left(1 - \frac{I_t}{n}\right)^{Z_{t+1}} = 1 - e^{-\Omega(\log\log n)} = 1 - o(1).$$

Thus $\mathbf{Pr}\left[X_u = 1\right] > \frac{3}{4}$ and $\mathbf{E}\left[X\right] = \sum_{u\in\mathcal{L}'(Z_{t+1})} \mathbf{Pr}\left[X_u = 1\right] > \frac{3}{4}|\mathcal{L}'(Z_{t+1})|$. Since $|\mathcal{L}'(Z_{t+1})| = |\mathcal{L}(Z_{t+1}) \cap \mathcal{U}_{t+1}| > \frac{3}{4}L(Z_{t+1})$, $\mathbf{E}\left[X\right] \geqslant \frac{9}{16}L(Z_{t+1})$. We know that $I_t \geqslant n^{\frac{1}{\beta-1}}/(2\log n)$ and also I_t is a non-decreasing function in t, so

$$Z_{t+1} = \frac{n\log\log n}{I_t} \leqslant 2\cdot n^{\frac{\beta-2}{\beta-1}}\log n\log\log n < n^{\frac{1}{\beta-1}}/\log n,$$

where the last inequality holds because $\beta < 3$. Now we can apply Lemma 4.3 (see appendix) to infer that with probability $1 - o(\frac{1}{n})$,

$$L(Z_{t+1}) \geqslant \frac{n\cdot c_1\cdot Z_{t+1}^{1-\beta}}{2} \geqslant \frac{c_1\cdot \log^{\beta-1} n}{2}.$$

Therefore, $\mathbf{E}\left[X\right] \geqslant \frac{9\cdot c_1\cdot\log^{\beta-1} n}{32}$. Then applying a Chernoff bound results into

$$\mathbf{Pr}\left[X < \frac{\mathbf{E}\left[X\right]}{2}\right] \leqslant \mathbf{Pr}\left[|X - \mathbf{E}\left[X\right]| \geqslant \frac{\mathbf{E}\left[X\right]}{2}\right] < 2e^{-\frac{\mathbf{E}[X]}{10}} \leqslant 2e^{-\Omega(\log^{\beta-1} n)}.$$

So with probability $1 - o(\frac{1}{n})$, we have that

$$|\mathcal{L}(Z_{t+1}) \cap \mathcal{I}_{t+1}| \geqslant X \geqslant \frac{\mathbf{E}\left[X\right]}{2} > \frac{3|\mathcal{L}'(Z_{t+1})|}{8} \geqslant \frac{1}{4}L(Z_{t+1}),$$

where the last inequality holds because $|\mathcal{L}'(Z_{t+1})| > \frac{3}{4}L(Z_{t+1})$. □

Lemma 4.7. *With probability* $1 - o(1)$*,* $T_2 - T_1 = \mathcal{O}(\log\log n)$*.*

Proof. Since $I_t \geqslant n^{\frac{1}{\beta-1}}/(2\log n)$, $Z_{t+1} = \frac{n\log\log n}{I_t} < n^{\frac{1}{\beta-1}}/\log n$, using Lemma 4.6 results into a lower bound for $|\mathcal{L}(Z_{t+1}) \cap \mathcal{I}_{t+1}|$. So with probability $1 - o(\frac{1}{n})$,

$$S_{t+1} = \sum_{u\in I_{t+1}} C_u \geqslant |\mathcal{L}(Z_{t+1} \cap \mathcal{I}_{t+1})| \cdot Z_{t+1} \geqslant \frac{1}{4}L(Z_{t+1})\cdot Z_{t+1}.$$

By applying Lemma 4.3, we conclude that with probability $1 - o(\frac{1}{n})$, $L(Z_{t+1}) \geqslant \frac{n\cdot c_1\cdot Z_{t+1}^{1-\beta}}{2}$. Therefore, with probability $1 - o(\frac{1}{n})$, $S_{t+1} \geqslant \frac{n\cdot c_1\cdot Z_{t+1}^{2-\beta}}{8}$. As long as $S_{t+1} = o(n)$, we can apply Lemma 4.2 for the Push protocol to round $t+2$ implying that with probability $1 - o(\frac{1}{\log n})$,

$$I_{t+2} = I_{t+1} + N_t \geqslant I_{t+1} + S_{t+1}(1 - o(1)).$$

Thus,

$$I_{t+2} > \frac{S_{t+1}}{2} \geqslant \frac{c_1}{16} n \cdot Z_{t+1}^{2-\beta} = \frac{c_1}{16} \cdot n^{3-\beta} \cdot \log\log^{2-\beta} n \cdot I_t^{\beta-2}.$$

By an inductive argument, we obtain that for any integer $k \geqslant 1$ with $S_{t+k} = o(n)$, it holds with probability $1 - o(\frac{k}{\log n})$,

$$\begin{aligned} I_{t+2k} &> \left(\frac{c}{16} n^{3-\beta} \cdot \log\log^{2-\beta} n\right)^{\sum_{i=0}^{k-1} (\beta-2)^i} I_t^{(\beta-2)^k} \\ &= \left(\frac{c}{16} n^{3-\beta} \cdot \log\log^{2-\beta} n\right)^{\frac{1-(\beta-2)^k}{3-\beta}} I_t^{(\beta-2)^k}. \end{aligned}$$

Therefore there exists $k = \mathcal{O}(\log_{\frac{1}{\beta-2}} \log n)$ such that

$$\begin{aligned} I_{t+2k} &\geqslant \left(\frac{c}{16} n^{3-\beta} \cdot \log\log^{2-\beta} n\right)^{\frac{1-\mathcal{O}(1/\log n)}{3-\beta}} I_t^{1/\log n} \\ &= \Omega\left(n^{1-\mathcal{O}(1/\log n)} \left(\frac{c}{16} \cdot \log\log^{2-\beta} n\right)^{\frac{1-\mathcal{O}(1/\log n)}{3-\beta}} \right) = \Omega\left(\frac{n}{\log\log^{\delta} n}\right), \end{aligned}$$

where $\delta = \frac{\beta-2}{3-\beta}(1 - \mathcal{O}(1/\log n)) > 0$. Hence $T_2 \leqslant T_1 + 2k = T_1 + \mathcal{O}(\log\log n)$ with probability $1 - o(1)$. □

The Final Phase. This phase starts with at least $\frac{n}{\log n}$ informed nodes. Since the runtime of our Push-Pull protocol is stochastically smaller than the runtime of the standard Push-Pull protocol (i.e. $C_u = 1$ for every $u \in V$), we simply use the result by Karp et. al in [18, Theorem 2.1] for the standard Push-Pull protocol which states that once $I_t \geqslant \frac{n}{\log n}$, additional $\mathcal{O}(\log\log n)$ rounds are with probability $1 - o(1)$ sufficient to inform all n nodes.

References

1. Berenbrink, P., Elsässer, R., Friedetzky, T.: Efficient randomised broadcasting in random regular networks with applications in peer-to-peer systems. In: Proc. 27th Symp. Principles of Distributed Computing (PODC), pp. 155–164 (2008)
2. Berger, N., Borgs, C., Chayes, J., Saberi, A.: On the Spread of Viruses on the Internet. In: Proc. 16th Symp. Discrete Algorithms (SODA), pp. 301–310 (2005)
3. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Transactions on Information Theory 52(6), 2508–2530 (2006)
4. Censor-Hillel, K., Shachnai, H.: Partial information spreading with application to distributed maximum coverage. In: Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, pp. 161–170 (2010)
5. Chierichetti, F., Lattanzi, S., Panconesi, A.: Almost tight bounds for rumour spreading with conductance. In: Proc. 42nd Symp. Theory of Computing (STOC), pp. 399–408 (2010)
6. Chung, F.R.K., Lu, L.: The average distance in a random graph with given expected degrees. Internet Mathematics 1(1), 91–113 (2003)

7. Clementi, A.E.F., Pasquale, F., Silvestri, R.: Opportunistic manets: Mobility can make up for low transmission power. IEEE/ACM Trans. Netw. 21(2), 610–620 (2013)
8. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proc. 6th Symp. Principles of Distributed Computing (PODC), pp. 1–12 (1987)
9. Doerr, B., Fouz, M.: Asymptotically optimal randomized rumor spreading. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 502–513. Springer, Heidelberg (2011)
10. Doerr, B., Fouz, M., Friedrich, T.: Social networks spread rumors in sublogarithmic time. In: Proc. 43rd Symp. Theory of Computing (STOC), pp. 21–30 (2011)
11. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. Random Structures and Algorithms 1(4), 447–460 (1990)
12. Fountoulakis, N., Huber, A., Panagiotou, K.: Reliable broadcasting in random networks and the effect of density. In: Proc. 29th IEEE Conf. Computer Communications (INFOCOM), pp. 2552–2560 (2010)
13. Fountoulakis, N., Panagiotou, K., Sauerwald, T.: Ultra-fast rumor spreading in social networks. In: Proc. 23rd Symp. Discrete Algorithms (SODA), pp. 1642–1660 (2012)
14. Frieze, A., Grimmett, G.: The shortest-path problem for graphs with random-arc-lengths. Discrete Applied Mathematics 10, 57–77 (1985)
15. Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: Proc. 28th Symp. Theoretical Aspects of Computer Science (STACS), pp. 57–68 (2011)
16. Giakkoupis, G.: Tight bounds for rumor spreading with vertex expansion. In: Proc. 25th Symp. Discrete Algorithms, SODA (to appear, 2014)
17. Harchol-Balter, M., Leighton, F.T., Lewin, D.: Resource discovery in distributed networks. In: Proc. 18th Symp. Principles of Distributed Computing (PODC), pp. 229–237 (1999)
18. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized Rumor Spreading. In: Proc. 41st Symp. Foundations of Computer Science (FOCS), pp. 565–574 (2000)
19. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. 44th Symp. Foundations of Computer Science (FOCS), pp. 482–491 (2003)
20. Mosk-Aoyama, D., Shah, D.: Fast distributed algorithms for computing separable functions. IEEE Transactions on Information Theory 54(7), 2997–3007 (2008)
21. Peres, Y., Sinclair, A., Sousi, P., Stauffer, A.: Mobile geometric graphs: Detection, coverage and percolation. In: Proc. 22nd Symp. Discrete Algorithms (SODA), pp. 412–428 (2011)
22. Pittel, B.: On spreading a rumor. SIAM Journal on Applied Mathematics 47(1), 213–223 (1987)
23. van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. In: Proceedings of the 15th IFIP International Conference on Distributed Systems Platforms, Middleware 1998, pp. 55–70 (1998)

Approximating the Value of a Concurrent Reachability Game in the Polynomial Time Hierarchy[⋆]

Søren Kristoffer Stiil Frederiksen and Peter Bro Miltersen

Aarhus University

Abstract. We show that the value of a finite-state concurrent reachability game can be approximated to arbitrary precision in $\mathsf{TFNP}[\mathsf{NP}]$, that is, in the polynomial time hierarchy. Previously, no better bound than PSPACE was known for this problem. The proof is based on formulating a variant of the state reduction algorithm for Markov chains using arbitrary precision floating point arithmetic and giving a rigorous error analysis of the algorithm.

1 Introduction

A *concurrent reachability game* (e.g., [3,1,8,7]) G is a finitely presented two-player game of potentially infinite duration, played between Player 1, the *reachability* player, and Player 2, the *safety* player. The arena of the game consists of a finite set of positions $0, 1, 2, \ldots, N$. When play begins, a pebble rests at position 1, the "start position". At each stage of play, with the pebble resting at a particular "current" position k, Player 1 chooses an *action* $i \in \{1, 2, \ldots, m\}$ while Player 2 concurrently, and without knowledge of the choice of Player 1 similarly chooses an action $j \in \{1, 2, \ldots, m\}$. A fixed and commonly known *transition function* $\pi : \{1, 2, \ldots, N\} \times \{1, 2, \ldots, m\}^2 \rightarrow \{0, 1, 2, \ldots, N\}$ determines the next position of the pebble, namely $\pi(k, i, j)$. If the pebble ever reaches 0 (the "goal position"), play ends, and Player 1 wins the game. If the pebble *never* reaches goal, Player 2 wins.

A *stationary strategy* for a player is a family of probability distribution on his actions, one for each state of the game. Everett [5] showed that every concurrent reachability game has a *value* which is a real number $v \in [0, 1]$ with the following properties [5,12,9]:

– For every $\epsilon > 0$, the reachability player has a stationary strategy for playing the game that guarantees that the pebble *eventually* reaches position 0 with probability at least $v - \epsilon$, no matter what the safety player does; such a strategy is called ϵ-optimal.

⋆ The authors acknowledge support from The Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

– The safety player has a stationary strategy for playing the game that guarantees that the pebble will never reach goal with probability at least $1-v$, no matter what the reachability player does; such a strategy is called optimal.

The present paper concerns the computation of v, given an explicit representation of the game (by its transition function). More precisely, as v can be an irrational number, we consider finding an approximation to v within an additive error of $\epsilon > 0$, when the transition function and ϵ (in standard fixed point binary representation, e.g., 0.00001) are given as input. This problem has an interesting history: Chatterjee *et al.* [2] claimed that the problem is in $\mathsf{NP} \cap \mathsf{coNP}$. Their suggested nondeterministic algorithm supposedly establishing this result was based on guessing stationary strategies for the two players. Etessami and Yannakakis [4] pointed out that the correctness proof of the algorithm of Chatterjee *et al.* is not correct, and that the best known upper bound on the complexity of the problem remained to be PSPACE, a bound that follows from a reduction to the decision problem for first order theory of the real numbers. The crucial flaw in the argument of Chatterjee *et al.* was its failure to establish correctly that the length of the standard fixed point bit representation of the numbers associated with the stationary strategies to be guessed is polynomially bounded in the size of the input. Hansen, Koucky and Miltersen [8] subsequently established that some games actually require strategies whose standard fixed point bit representations have superpolynomial size. That is, not only was the correctness proof of Chatterjee *et al.* incorrect, but so was the algorithm itself.

The main result of the present paper is the first "complexity class upper bound" better than PSPACE on the computational complexity of the problem of approximating the value of a concurrent reachability game. More specifically, consider the search problem APPROX-CRG-VALUE which on input $\langle G, 1^k \rangle$ finds an approximation to the value of the finite state concurrent reachability game G within additive error 2^{-k}. Then, our main theorem is the following.

Theorem 1. *APPROX-CRG-VALUE can be solved in* $\mathsf{TFNP[NP]}$

The class $\mathsf{TFNP[NP]}$ ("total functions from NP with an oracle for NP") was defined by Megiddo and Papadimitriou [10]. A total search problem can be solved in this class if there is a nondeterministic Turing machine M with an oracle for an NP language, so that M runs in polynomial time and on all computation paths either outputs *fail* or a correct solution to the input (in this case, a value approximation), and on at least one computation path does the latter. For readers unfamiliar with (multi-valued) search problem classes, we point out that by a standard argument, the fact that APPROX-CRG-VALUE is in $\mathsf{TFNP[NP]}$ implies that there is a *language* L in $\Delta_3^p = \mathsf{P[NP[NP]]}$ encoding a (single-valued) function f, so that $f(G, 1^k)$ approximates the value of G within additive error 2^{-k}.

Interestingly, the main key to establishing our result is to work with *floating point* rather with *fixed point* representation of the real numbers involved in the computation. We are not aware of any previous case where this distinction has been important for establishing membership in a complexity class. Nevertheless, it is natural that this distinction turns out to be important in the context of concurrent reachability games as good strategies in those are known to involve real numbers of very different magnitude (such as 2^{-1} and $2^{-10000000}$), by the examples of Hansen, Koucky and Miltersen. As the main technical tool, we adapt the *state reduction algorithm* for analyzing Markov chains due to Sheskin [13] and Grassman *et al.* [6]. This algorithm was shown to have very good numerical stability by O'Cinneide [11] in contrast to the standard ways of analyzing Markov chains using matrix inversion. Our adapted finite precision algorithm computes absorption probabilites rather than steady state probabilities and the numerical stability argument of O'Cinneide is adapted so that a formal statement concerning polytime Turing machine computations on arbitrary precision floating point numbers, with numbers of widely different orders of magintude appearing in a single computation, is obtained (Theorem 4 below). We emphasize that the adaptation is standard in the context of numerical analysis – in particular, the error analysis is an instance of the backward error analysis paradigm due to Wilkinson [15] – but to the best of our knowledge, the bridge to formal models of computation and complexity classes was not previously built.

2 Preliminaries

Relative Distance and Closeness

For non-negative real numbers $x, \tilde{x}$, we define the *relative distance between* $\tilde{x}$ *and* x to be $\delta(x, \tilde{x}) = \frac{\max(x,\tilde{x})}{\min(x,\tilde{x})} - 1$ with the convention that $0/0 = 1$ and $c/0 = +\infty$ for $c > 0$. We shall say that a non-negative real x is (u, j)-*close* to a non-negative real $\tilde{x}$ where u and j are non-negative integers if $\delta(x, \tilde{x}) \leq (\frac{1}{1-2^{-u+1}})^j - 1$. We omit the proofs of the following straightforward lemmas.

Lemma 1. *If x is (u, i)-close to y and y is (u, j)-close to z, then x is $(u, i+j)$-close to z.*

Lemma 2. *Let $x, \tilde{x}, y, \tilde{y}$ be non-negative real numbers so that x is (u, i)-close to $\tilde{x}$ and y is (u, j)-close to $\tilde{y}$. Then, $x + y$ is $(u, \max(i, j))$-close to $\tilde{x} + \tilde{y}$, xy is $(u, i + j)$-close to $\tilde{x}\tilde{y}$ and x/y is $(u, i + j)$-close to $\tilde{x}/\tilde{y}$.*

Floating Point Numbers

Let $\mathbb{D}(u)$ denote the set of non-negative dyadic rationals with a u-bit mantissa, i.e.

$$\mathbb{D}(u) = \{0\} \cup \{x2^{-i} | x \in \{2^{u-1}, 2^{u-1} + 1, \ldots, 2^u - 1\}, i \in \mathbf{Z}\}$$

The *u-bit floating point representation* of an element $x2^{-i} \in \mathbb{D}(u)$ is $\langle 1^u, b(x), b(i)\rangle$, where b denotes the map taking an integer to its binary representation. Note that the representation is unique (for fixed u). The *exponent* of $x2^{-i} \in \mathbb{D}(u)$ is the number $-i$; note that this is well-defined. The u-bit floating point representation of 0 is $\langle 1^u, 0\rangle$. For convenience of expression, we shall blur the distinction between an element of $\mathbb{D}(u)$ and its floating point representation. Let $\oplus^u, \oslash^u, \otimes^u$ denote the finite precision analogoues of the arithmetic operations $+, /, *$. All these operations map $\mathbb{D}(u)^2$ to $\mathbb{D}(u)$ and are defined by truncating (rounding down) the result of the corresponding exact arithmetic operation to u digits.

The following lemma is straightforward.

Lemma 3. *Let x be a non-negative real number and let u be a positive integer. There is a number $\tilde{x} \in \mathbb{D}(u)$, which is $(u,1)$-close to x.*

Lemma 4. *Let $\tilde{x}, \tilde{y} \in \mathbb{D}(u)$ with $\tilde{x}$ being (u,i)-close to a non-negative real number x and $\tilde{y}$ being (u,j)-close to a non-negative real number y. Then $\tilde{x} \oplus^u \tilde{y}$ is $(u, \max(i,j)+1)$-close to $x+y$, $\tilde{x} \otimes^u \tilde{y}$ is $(u, i+j+1)$-close to xy and $\tilde{x} \oslash^u \tilde{y}$ is $(u, i+j+1)$-close to x/y.*

Proof. The statement follows from combining Lemma 2 and Lemma 1 and noting that we have that $(a+b)(1-2^{-u+1}) \leq a \oplus^u b \leq a+b$, and similarly for the other operations.

For technical reasons, we want to be able to represent probability distributions in floating point representation, i.e., as finite strings, in such a way that the semantics of each string is some exact, well-defined, actual probability distribution that we can refer to. Simply representing each probability in floating point will not work for us, as we would not be able to ensure the numbers summing up to exactly one. Therefore we adopt the following definition: We let $\mathbb{P}(u)$ denote the set of finite probability density functions $(p_1, p_2, \ldots, p_k)$ for some finite k with the property that there exists numbers $p'_1, \ldots, p'_k \in \mathbb{D}(u)$, so that $p_i = p'_i / \sum_j p'_j$ for $i = 1, .., k$ and so that $\sum_j p'_j$ is (u,k)-close to 1. We also refer to the vector $(p'_1, p'_2, \ldots, p'_k)$ as an *approximately normalized representation* of $(p_1, p_2, \ldots, p_k)$. As an example, $(\frac{1}{1+2^{-100}}, \frac{2^{-100}}{1+2^{-100}})$ is in $\mathbb{P}(64)$, and has an approximately normalized 64-bit floating point representation being the concatenation of the 64-bit floating point representations of the numbers 1 and 2^{-100} . On the other hand, $(1-2^{-100}, 2^{-100})$ is not in $\mathbb{P}(u)$ for any $u \leq 2^{100}$. The following lemma simply expresses that one can generate an approximately normalized floating point approximation of any unnormalized distribution by normalizing it numerically.

Lemma 5. *Let $a_1, a_2, \ldots, a_k \in \mathbb{D}(u)$ and let $\tilde{p}_i = a_i \oslash^u \left(\bigoplus^u_{j=1\ldots k} a_j\right)$. Also, let $p_i = \tilde{p}_i / \sum_{j=1..k} \tilde{p}_j$. Then $(p_1, \ldots, p_k) \in \mathbb{P}(u)$ and $(\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_k)$ is an approximately normalized representation of this distribution. Also, p_i is $(u, 2k)$-close to $a_i / \sum_j a_j$.*

Proof. By repeated use of Lemma 4, for each i, $a_i / \sum_j a_j$ is (u,k)-close to $\tilde{p}_i$. Therefore, by Lemma 2, we have that 1 is (u,k)-close to $\sum_i \tilde{p}_i$. Therefore, $(\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_k)$ satisfies the condition for being an approximately normalized representation. Also $p_i = \tilde{p}_i / \sum_j \tilde{p}_j$ is (u,k)-close to $\tilde{p}_i = \tilde{p}_i / 1$ by Lemma 2. Then, by Lemma 1, p_i is $(u, 2k)$ close to $a_i / \sum_j a_j$.

The following lemma expresses that every probability density function is well-approximated by an element of $\mathbb{P}(u)$.

Lemma 6. *Let $q = (q_1, q_2, \ldots, q_k)$ be a probability density function. There exists $p = (p_1, p_2, \ldots, p_k)$ in $\mathbb{P}(u)$ so that for all i, p_i and q_i are $(u, 2k+2)$-close.*

Proof. For $i = 1, \ldots, k$, let a_i be a number in $\mathbb{D}(u)$ which is $(u, 1)$-close to q_i, as guaranteed by Lemma 3. By Lemma 2, $\sum_i a_i$ is $(u, 1)$-close to $\sum_i q_i = 1$. Therefore, $a_i / \sum_j a_j$ is $(u, 2)$-close to q_i, again by Lemma 2. Now let p_i be the distribution in $\mathbb{P}(u)$ defined by applying Lemma 5 to (a_i). The statement of the lemma gives us that p_i is $(u, 2k)$-close to $a_i / \sum_j a_j$ which is $(u, 2)$-close to q_i, so by Lemma 1, p_i is $(u, 2k + 2)$-close to q_i.

Absorbing Markov Chains and Concurrent Reachability Games

An *absorbing Markov chain* is given by a finite set of transient states $\{1, \ldots, N\}$ and a finite set of absorbing states $\{N+1, \ldots, N+S\}$ and transition probabilities $p_{ij}, i \in \{1, \ldots, N\}, j \in \{1, \ldots, N + S\}$ with the property that for each transient state k_0, there are states $k_1, k_2, \ldots, k_l$ so that $p_{k_i k_{i+1}} > 0$ for all i and so that k_l is absorbing. We say that the chain is *loop-free* if $p_{ii} = 0$ for all $i \in \{1, \ldots, N\}$. Given an absorbing Markov chain, the *absorption probability* a_{ij} where i is transient and j is absorbing, is the probability that the chain is eventually absorbed in state j, given that it is started in state i.

We shall use the following theorem of Solan [14, Theorem 6] stating that the absorption probabilities of a Markov chain only change slightly when transition probabilities are perturbed (Solan's theorem is actually much more general; the statement below is its specialization to absorbing Markov chains).

Theorem 2. *Let M and $\tilde{M}$ be absorbing Markov chains with identical sets of transient states $\{1, 2, \ldots, N\}$ and absorbing states $\{N + 1, \ldots, N + S\}$ and transition probabilities p_{kl}, $\tilde{p}_{kl}$ respectively. Assume that for all $k, l \in \{1, \ldots, N\}$ we have $\delta(p_{kl}, \tilde{p}_{kl}) \leq \epsilon$. Let $a_{kl}, \tilde{a}_{kl}$ denote the absorption probabilities in the two chains. Then, for each $k \in \{1, \ldots, N\}$ and each $l \in \{N+1, \ldots, N+S\}$, we have $|a_{kl} - \tilde{a}_{kl}| \leq 4N\epsilon$.*

The formalities concerning concurrent reachability games were given in the introduction. We shall use the following theorem of Hansen, Koucky and Miltersen [8, Theorem 4].

Theorem 3. *For any concurrent reachability games with a total number of $A \geq 10$ actions in the entire game (collecting actions in all positions belonging to both players), and any $0 < \varepsilon < \frac{1}{2}$, Player 1 has an ϵ-optimal stationary strategy with all non-zero probabilities involved being at least $\varepsilon^{2^{30A}}$.*

3 The State Reduction Algorithm

In this section, we present an adaptation of the state reduction algorithm of Sheskin [13] and Grassman *et al.* [6] for computing steady-state probabilities in Markov chains. The algorithm is (straightforwardly) adapted to compute absorption probabilities instead of steady-state probabilities. Also, we adapt an analysis due to O'Cinneide [11] for the finite precision version of the algorithm. The adaptation is presented as a "theory of computation" flavored statement as Theorem 4.

Lemma 7. *There is a polynomial time algorithm MAKE-LOOP-FREE that*

- *takes as input the transition probability matrix of an absorbing Markov chain M with N transient states $\{1, 2, .., N\}$ and S absorbing states $\{N+1, .., N+S\}$, with each transition probability distribution of M being in $\mathbb{P}(u)$ and being given by an approximately normalized representation using u-bit floating point numbers, for an arbitrary $u \geq 1000(N+S)^2$,*
- *outputs the transition probability matrix of an absorbing loop-free Markov chain M' with N transient states $\{1, .., N\}$ and S absorbing states $\{N+1, .., N+S\}$ and with each transition probability distribution of M' being in $\mathbb{P}(u)$ and being represented by an approximately normalized representation using u-bit floating point numbers,*
- *with the smallest (negative) exponent among all floating point numbers in the output being at most one smaller than the smallest (negative) exponent among all floating point numbers in the input,*
- *and with the property that each absorption probability a_{ij} of M (for $i = 2 \dots N, j = N+1 \dots N+S$) differs from the corresponding absorption probability a'_{ij} of M' by at most $20(N+S)^3 2^{-u}$.*

Proof. We assume $N \geq 1$ and $S \geq 2$, otherwise the problem is trivial. Let the transition probabilities of the chain M be denoted p_{ij}. From each outcome of M seen as a sequence of states, consider removing all repeated occurrences of states (e.g., "2 2 6 6 6 5 5 .." becomes "2 6 5.."). This induces a probability distribution on sequences which is easily seen to be the distribution generated by a loop-free Markov chain $\bar{M}$ with transition probabilities q_{ij} for $i = 1, \dots, N, j = 1, \dots S+N, i \neq j$, with $q_{ij} = p_{ij}/q_i$ where $q_i = \sum_{k \neq i} p_{ik}$. Clearly, $\bar{M}$ has the same absorption probabilities as M. The algorithm MAKE-LOOP-FREE constructs an approximation M' to $\bar{M}$ as indicated by the pseudocode.

By Lemma 5, the output is a family of approximately normalized floating point representations of probability distributions $q'_{ij} = \tilde{q}_{ij}/\sum_k \tilde{q}_{ik}$. Let M' be the Markov chain M' with transition probabilities q'_{ij}. We need to show that M' has absorption probabilities close to the absorption probabilties of the chain M. For this, we need to bound the relative distance between q'_{ij} and q_{ij}. For this, note that:

- (i) Each $\tilde{p}_{ij}$ is $(u, N+S)$-close to p_{ij} by definition of approximately normalized representation.
- (ii) Each $\tilde{q}_i$ is $(u, 2N+2S-2)$-close to q_i by (i) and $N+S-2$ applications of Lemma 4.
- (iii) Each $\tilde{q}_{ij}$ is $(u, 2(N+S)(N+S-1))$-close to q_{ij} by (ii) and Lemma 4.
- (iv) Each q'_{ij} is $(u, N+S-1)$-close to $\tilde{q}_{ij}$ by Lemma 5 and the definition of approximately normalized representation.
- (v) Each q'_{ij} is $(u, 2(N+S)(N+S-1)+N+S-1)$-close to q_{ij} by (iii), (iv) and Lemma 1, i.e. at least $(u, 2(N+S)^2)$-close.

Theorem 2 now implies that the absorption probabilities of M' differ from the corresponding absorption probabilities of M by at most $4N\epsilon$ where $\epsilon =$

Algorithm 1. MAKE-LOOP-FREE

```
**Input:** $(\tilde{p}_{ij})_{i\in\{1,\ldots,N\},j\in\{1,\ldots,N+S\}}$, where for each $i = 1,\ldots,N$, $(\tilde{p}_{ij})_{j\in\{1,\ldots,N\}}$ is an approximately normalized $u$-bit floating point representation of a probability distribution $p_i$.
for $i = 1 \to N$ do
    $\tilde{q}_i \leftarrow \bigoplus^u_{l\in\{1,\ldots,N+S\}\setminus\{i\}} \tilde{p}_{il}$
    for $j = 1 \to N+S$ do
        if $i = j$ then
            $\tilde{q}_{ij} \leftarrow 0$
        else
            $\tilde{q}_{ij} \leftarrow \tilde{p}_{ij} \oslash^u \tilde{q}_i$
        end if
    end for
end for
return $(\tilde{q}_{ij})_{i\in\{1,\ldots,N\},j\in\{1,\ldots,N+S\}}$
```

$(\frac{1}{1-2^{-u+1}})^{2(N+2)^2} - 1$. Since $u \geq 1000(N+S)^2$, we have $(\frac{1}{1-2^{-u+1}})^{2(N+S)^2} \leq 1 + 5(N+S)^2 2^{-u}$, so $\epsilon \leq 5(N+S)^2 2^{-u}$, and $4N\epsilon \leq 20(N+S)^3 2^{-u}$. Finally, to show that the exponents in the output are at most one smaller than the exponents in the input, note that we actually have that $q_{ij} \geq p_{ij}$. Since $\tilde{q}_{ij}$ closely approximates q_{ij} and $\tilde{p}_{ij}$ closely approximates p_{ij}, it is not possible for $\tilde{q}_{ij}$ to be smaller than $\tilde{q}_{ij}$ by a factor of more than two, from which the claim follows.

Lemma 8. *There is a polynomial time algorithm APPROX-STATE-RED that*

– *takes as input the transition probability matrix of an absorbing loop-free Markov chain M with N transient states $\{1, 2, .., N\}$ and S absorbing states $\{N+1, .., N+S\}$, with each transition probability distribution of M being in $\mathbb{P}(u)$ and being given by an approximately normalized representation using u-bit floating point numbers, for an arbitrary $u \geq 1000(N+S+1)^2$,*
– *outputs the transition probability matrix of an absorbing loop-free Markov chain M' with $N-1$ transient states $\{2, .., N\}$ and S absorbing states $\{N+1, .., N+S\}$ and with each transition probability distribution of M' being in $\mathbb{P}(u)$ and being represented by an approximately normalized representation using u-bit floating point numbers,*
– *with the smallest (negative) exponent among all floating point numbers in the output being at most one smaller than the smallest (negative) exponent among all floating point numbers in the input,*
– *and with the property that each absorption probability a_{ij} of M (for $i = 2\ldots N, j = N+1\ldots N+S$) differs from the corresponding absorption probability a'_{ij} of M' by at most $80(N+S)^3 2^{-u}$.*

Proof. Let the transition probabilities of the chain M be denoted p_{ij}. From each outcome of M as a sequence of states, consider removing all occurrences of the state 1. This induces a probability distribution on sequences which is easily seen (recalling that M is loop free) to be the distribution generated by a Markov chain $\tilde{M}$ with transition probabilities q_{ij}, for $i, j = 2, .., N$ with $q_{ij} = p_{ij} + p_{i1}p_{1j}$. Clearly, $\tilde{M}$ has the same absorption probabilities as M. From each outcome of $\tilde{M}$ since as a sequence of states, consider removing all repeated occurrences of states (e.g., "2 2 6 6 6 5 5 .." becomes "2 6 5.."). This induces a probability distribution on sequences which is easily seen to be the distribution generated by a loop-free Markov chain $\bar{M}$ with transition probabilities r_{ij} for $i, j = 2, \ldots, N, i \neq j$, with $r_{ij} = q_{ij}/q_i$ where $q_i = \sum_{k \neq i} q_{ik}$. Clearly, $\bar{M}$ has the same absorption probabilities as $\tilde{M}$, and hence as M. The algorithm APPROX-STATE-RED constructs an approximation M' to $\bar{M}$ as indicated by the pseudocode.

Algorithm 2. APPROX-STATE-RED

```
Input: $(\tilde{p}_{ij})_{i \in \{1,\ldots,N\}, j \in \{1,\ldots,N+S\}}$, where for each $i = 1, \ldots, N$, $(\tilde{p}_{ij})_{j \in \{1,\ldots,N\}}$ is
an approximately normalized $u$-bit floating point representation of a probability
distribution $p_i$, and $\tilde{p}_{ii} = 0$.
for $i, j = 2 \to N$ do
    if $i = j$ then
        $\tilde{q}_{ij} \leftarrow 0$
    else
        $\tilde{q}_{ij} \leftarrow \tilde{p}_{ij} \oplus^u (\tilde{p}_{i,1} \otimes^u \tilde{p}_{1,j})$
    end if
end for
for $i = 2 \to N$ do
    $\tilde{q}_i \leftarrow \bigoplus^u_{l \in \{2,\ldots,N+S\} \setminus \{i\}} \tilde{q}_{il}$
    for $j = 2 \to N + S$ do
        if $i = j$ then
            $\tilde{r}_{ij} \leftarrow 0$
        else
            $\tilde{r}_{ij} \leftarrow \tilde{q}_{ij} \oslash^u \tilde{q}_i$
        end if
    end for
end for
return $(\tilde{r}_{ij})_{i \in \{1,\ldots,N\} \setminus \{1\}, j \in \{1,\ldots,N+S\} \setminus \{1\}}$
```

By Lemma 5, the output is a family of approximately normalized floating point representations of probability distributions $r'_{ij} = \tilde{r}_{ij} / \sum_k \tilde{r}_{ik}$. Let M' be the Markov chain M' with transition probabilities r'_{ij}. We need to show that M' has absorption probabilities close to the absorption probabilties of the chain M. For this, we need to bound the relative distance between r'_{ij} and r_{ij}. For this, note that:

– (i) Each $\tilde{p}_{ij}$ is $(u, N + S)$-close to p_{ij} by definition of approximately normalized representation.
– (ii) Each $\tilde{q}_{ij}$ is $(u, 2(N + S) + 2)$-close to q_{ij} by (i) and two applications of Lemma 4.
– (iii) Each $\tilde{q}_i$ is $(u, 3(N + S) - 1)$-close to q_i by (ii) and $N + S - 3$ applications of Lemma 4.

- (iv) Each $\tilde{r}_{ij}$ is $(u, 8(N+S)^2)$-close to r_{ij} by (iii) and Lemma 4.
- (v) Each r'_{ij} is $(u, N+S-1)$-close to $\tilde{r}_{ij}$ by Lemma 5 and the definition of approximately normalized representation.
- (vi) Each r'_{ij} is $(u, 9(N+S)^2)$-close to r_{ij} by (iv), (v) and Lemma 1.

Theorem 2 now implies that the absorption probabilities of M' differ from the corresponding absorption probabilities of M by at most $4N\epsilon$ where $\epsilon = (\frac{1}{1-2^{-u+1}})^{9(N+S)^2} - 1$. Since $u \geq 1000(N+S+1)^2$, we have $(\frac{1}{1-2^{-u+1}})^{9(N+S)^2} \leq 1 + 20(N+S)^2 2^{-u}$, so $\epsilon \leq 20(N+S)^2 \cdot 2^{-u}$, and $4N\epsilon \leq 80(N+S)^3 2^{-u}$. Finally, to show that the exponents in the output are at most one smaller than the exponents in the input, note that we actually have that $r_{ij} \geq p_{ij}$. Since $\tilde{r}_{ij}$ closely approximates r_{ij} and $\tilde{p}_{ij}$ closely approximates p_{ij}, it is not possible for $\tilde{r}_{ij}$ to be smaller than $\tilde{p}_{ij}$ by a factor of more than two, from which the claim follows.

Theorem 4. *There is a polynomial time algorithm APPROX-ABSORPTION that takes as input the transition probability matrix of an absorbing Markov chain M with n states, with each transition probability distribution of M being in $\mathbb{P}(u)$ and being given by an approximately normalized representation using u-bit floating point numbers for some $u \geq 1000n^2$, and outputs for each transient state i and each absorbing state j, an approximation to the absorption probability a_{ij} given in u-bit floating point notation and with additive error at most* $80n^4 2^{-u}$.

Proof. For each absorption probability to be estimated, relabel states so that transient states are labeled 1,2,..N, with the transient state of interest being N. Then apply MAKE-LOOP-FREE of Lemma 7 once, and then APPROX-STATE-RED of Lemma 8 $N-1$ times, eliminating all transient states but the one of interests. As the final Markov chain is loop-free and has only one transient state, its absorption probabilities are equal to its transition probabilities and are approximations with the desired accuracies to the absorption probabilities of the orginal chain by the two lemmas.

4 Approximating Values in the Polynomial Time Hierarchy

In this section, we prove our main result, Theorem 1. Let L_1 be the language of tuples $\langle M, 1^u, \alpha\rangle$, where M is an absorbing Markov chain with two absorbing states *goal* and *trap* and a distinguished start state *start*, the parameter u satisfies the conditions of Theorem 4, α is the standard fixed point binary notation of a number between 0 and 1 (which we will also call α), and when APPROX-ABSORPTION of Theorem 4 is applied to M, the approximation returned for the probability of being absorbed in *goal* when the chain is started in *start* is at least α. Then, since APPROX-ABSORPTION is a polynomial time algorithm, we have that $L_1 \in \mathsf{P}$. Given a concurrent reachability game G, if Player 1's strategy is fixed to x and Player 2's strategy is fixed to y, we get a Markov chain. If we collapse all states in this Markov chain from which the goal position 0 will be reached with probability 0 into a single state -1, we get an absorbing Markov chain with two absorbing states, 0 (*goal*) and -1 (*trap*). Let this Markov chain be denoted $M(G, x, y)$. Let L_2 be the language of tuples $\langle G, 1^u, i, x, \alpha\rangle$, so that G is a concurrent reachability game, i is either 1 or 2, x is a stationary strategy for Player i with each involved probability distribution being represented approximately normalized using u-bit floating point notation, and α is the standard fixed point binary representation of a number between 0 and 1, so that *for all* pure strategies y of Player $3 - i$, we have that if $i = 1$ then $\langle M(G, x, y), 1^u, \alpha\rangle \in L_1$ and if $i = 2$, then $\langle M(G, y, x), 1^u, \alpha\rangle \in L_1$. Then, by construction, and since a pure strategy y has a bit representation bounded in size by the bit representation of the game, $L_2 \in \mathsf{coNP}$.

We are now ready to show that APPROX-CRG-VALUE can be solved in $\mathsf{TFNP}[\mathsf{NP}]$ by presenting an appropriate Turing machine M. The machine M uses the language L_2 (or its complement, a language in NP) as its oracle and does the following on input $\langle G, 1^k\rangle$: Let N be the number of non-terminal positions of G and m the largest number of actions for a player in any state. Let $u^* = 1000km(N+2)^3$ and let $e^* = (k+4)2^{30A}+1$, where A is the maximum of 10 and the total number of actions in G, collecting in each state all actions of both players. The machine nondeterministically guesses an integer j between 0 and 2^{k+1} and a strategy profile (x, y) for the two players with each involved probability distribution being in $\mathbb{P}(u^*)$ and with probabilities having exponents at least $-e^*$ (note that e^* has polynomially many bits in the standard binary representation, so it is possible for M to do this). If $\langle G, 1^{u^*}, 1, x, (j-1)2^{-k-1}\rangle \in L_2$ and $\langle G, 1^{u^*}, 2, y, (j+1)2^{-k-1}\rangle \in L_2$ the machine outputs $j2^{-k-1}$, otherwise it outputs *fail.* We argue that M does the job correctly: Suppose M outputs a number $j2^{-k-1}$. In this case, M has guessed a strategy x for Player 1 and a strategy y for Player 2, so that $\langle G, 1^{u^*}, 1, x, (j-1)2^{-k-1}\rangle \in L_2$ and $\langle G, 1^{u^*}, 2, y, (j+1)2^{-k+1}\rangle \in L_2$. Such a strategy x guarantees that *goal* is reached with probability at least $(j-1)2^{-k-1}$ minus the additive error of the estimate provided by APPROX-ABSORPTION, that is, with probability larger that $(j-2)2^{-k-1}$. Similarly, the strategy y guarantees that *goal* is reached with probability at most $(j+2)2^{-k-1}$. So the value of the game is in the interval $[(j-2)2^{-k-1}, (j+2)2^{-k-1}]$ and the approximation $j2^{-k-1}$ is indeed 2^{-k}-accurate. Finally, we show that M does not output *fail* on all computation paths. Consider a path where M guesses j, where j is a number so that the value v of the game is in $[(2j-1)2^{-k-2}, (2j+1)2^{-k-2}]$. Let x^* be an 2^{-k-4}-optimal stationary strategy for Player 1 with all non-zero probabilities involved being bigger than $2^{-(k+4)2^{30A}}$, as guaranteed by Theorem 3. Let x be the stationary strategy that in each state is given by the probability distribution from $\mathbb{P}(u^*)$ obtained by applying Lemma 6 to the distribution in each state of x^*. The relative distance between probabilities according to x and probabilities according to x^* is at most $\gamma = 1/(1 - 2^{-u^*})^{2m+2}$, by that lemma. The exponents involved in an approximately normalized representation of x' are all at least $-e^*$, by x' closely approximating x^*. For each pure reply y of Player 2, Theorem 2 yields that the probability of the process being absorbed in *goal* in the chain $M(x', y)$ is at least $v - 2^{-k-4} - 4N\gamma \geq v - 2^{-k-3}$. When APPROX-ABSORPTION is applied to $M(x, y)$, its estimate for this probability has error much smaller than 2^{-k-3}, that is, its estimate is larger than $v - 2^{-k-2}$. That is, $\langle G, 1^{u^*}, 1, x, (j-1)2^{-k-1}\rangle \in L_2$. A similar construction yields a y so that $\langle G, 1^{u^*}, 2, y, (j+1)2^{-k-1}\rangle \in L_2$. If M guesses j, x, y, it does not output *fail*. This completes the proof.

References

1. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Strategy improvement for concurrent reachability games. In: Third International Conference on the Quantitative Evaluation of Systems, QEST 2006, pp. 291–300. IEEE Computer Society (2006)
2. Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)
3. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theor. Comput. Sci. 386(3), 188–217 (2007)
4. Etessami, K., Yannakakis, M.: Recursive concurrent stochastic games. Logical Methods in Computer Science 4(4) (2008)
5. Everett, H.: Recursive games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games Vol. III, vol. 39, Annals of Mathematical Studies. Princeton University Press (1957)
6. Grassmann, W.K., Taksar, M.I., Heyman, D.P.: Regenerative analysis and steady state distributions for Markov chains. Operations Research, 1107–1116 (1985)
7. Hansen, K.A., Ibsen-Jensen, R., Miltersen, P.B.: The complexity of solving reachability games using value and strategy iteration. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 77–90. Springer, Heidelberg (2011)
8. Hansen, K.A., Koucky, M., Miltersen, P.B.: Winning concurrent reachability games requires doubly exponential patience. In: 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp. 332–341. IEEE (2009)
9. Himmelberg, C.J., Parthasarathy, T., Raghavan, T.E.S., Vleck, F.S.V.: Existence of p-equilibrium and optimal stationary strategies in stochastic games. Proc. Amer. Math. Soc. 60, 245–251 (1976)
10. Megiddo, N., Papadimitriou, C.H.: On total functions, existence theorems and computational complexity. Theor. Comput. Sci. 81(2), 317–324 (1991)
11. O'Cinneide, C.A.: Entrywise perturbation theory and error analysis for Markov chains. Numerische Mathematik 65(1), 109–120 (1993)
12. Parthasarathy, T.: Discounted and positive stochastic games. Bull. Amer. Math. Soc. 77, 134–136 (1971)
13. Sheskin, T.J.: A Markov chain partitioning algorithm for computing steady state probabilities. Operations Research, 228–235 (1985)
14. Solan, E.: Continuity of the value of competitive Markov decision processes. Journal of Theoretical Probability 16, 831–845 (2003)
15. Wilkinson, J.H.: A priori error analysis of algebraic processes. In: Proceedings International Congress Math., pp. 629–639. Izdat. Mir, Moscow (1968)

Computing a Walrasian Equilibrium in Iterative Auctions with Multiple Differentiated Items

Kazuo Murota[1], Akiyoshi Shioura[2], and Zaifu Yang[3]

[1] University of Tokyo, Tokyo 113-8656, Japan
murota@mist.i.u-tokyo.ac.jp
[2] Tohoku University, Sendai 980-8579, Japan
shioura@dais.is.tohoku.ac.jp
[3] University of York, York YO10 5DD, UK
zaifu.yang@york.ac.uk

Abstract. We address the problem of computing a Walrasian equilibrium price in an ascending auction with gross substitutes valuations. In particular, an auction market is considered where there are multiple differentiated items and each item may have multiple units. Although the ascending auction is known to find an equilibrium price vector in finite time, little is known about its time complexity. The main aim of this paper is to analyze the time complexity of the ascending auction globally and locally, by utilizing the theory of discrete convex analysis. An exact bound on the number of iterations is given in terms of the ℓ_∞ distance between the initial price vector and an equilibrium, and an efficient algorithm to update a price vector is designed based on a min-max theorem for submodular function minimization.

1 Introduction

We study an ascending auction, where given a set of discrete (or indivisible) items, the auctioneer aims to find an efficient allocation of items to bidders as well as market clearing prices of the items (see [5,6] for surveys). In recent years, there has been a growing use of iterative auctions for items such as spectrum licenses in telecommunication, electrical power, landing slots at airports, etc. In this paper, we consider the setting where there are multiple indivisible items for sale and each item may have multiple units; this is more general than the single-unit setting used extensively in the literature. A fundamental concept in auctions is the Walrasian equilibrium (or competitive equilibrium), which is a pair of a price vector and an allocation of items satisfying a certain property (see below for the precise definition). The main aim of this paper is to analyze the problem of computing a Walrasian equilibrium with respect to the time complexity, by utilizing the theory of discrete convex analysis.

Multi-item Auction and Walrasian Equilibrium. The auction market model is formulated as follows. In the market, there are n types of items or goods, denoted by $N = \{1, 2, \ldots, n\}$, and m bidders, denoted by $M = \{1, 2, \ldots, m\}$.

We have $u(i) \in \mathbb{Z}_+$ units available for each item $i \in N$. The case with $u(i) = 1$ ($i \in N$) is referred to as the *single-unit auction* in this paper. We denote the integer interval as $[\mathbf{0}, u]_\mathbb{Z} = \{x \in \mathbb{Z}^n \mid \mathbf{0} \leq x \leq u\}$; note that $[\mathbf{0}, \mathbf{1}]_\mathbb{Z} = \{0, 1\}^n$. Each vector $x \in [\mathbf{0}, u]_\mathbb{Z}$ is often called a *bundle*; a bundle x corresponds to a (multi)-set of items, where $x(i)$ represents the multiplicity of item $i \in N$. Each bidder $j \in M$ has his valuation function $f_j : [\mathbf{0}, u]_\mathbb{Z} \to \mathbb{R}$; the value $f_j(x)$ represents the degree of satisfaction for a bundle x. An *allocation* of items is defined as a set of bundles $x_1, x_2, \ldots, x_m \in [\mathbf{0}, u]_\mathbb{Z}$ satisfying $\sum_{j=1}^m x_j = u$.

In an auction, we want to find an efficient allocation and market clearing prices. Given a price vector $p \in \mathbb{R}^n$, each bidder $j \in M$ wants to have a bundle x which maximizes the value $f_j(x) - p^\top x$. For $j \in M$ and $p \in \mathbb{R}^n$, define

$$V_j(p) = \max\{f_j(x) - p^\top x \mid x \in [\mathbf{0}, u]_\mathbb{Z}\}, \tag{1}$$

$$D_j(p) = \arg\max\{f_j(x) - p^\top x \mid x \in [\mathbf{0}, u]_\mathbb{Z}\}. \tag{2}$$

We call the function $V_j : \mathbb{R}^n \to \mathbb{R}$ and the set $D_j(p) \subseteq [\mathbf{0}, u]_\mathbb{Z}$ an *indirect utility function* and a *demand set*, respectively. On the other hand, the auctioneer wants to find a price vector under which all items are sold. Hence, all of the auctioneer and bidders are happy if we can find a pair of a price vector p^* and an allocation $x_1^*, x_2^*, \ldots, x_m^*$ satisfying the condition that $x_j^* \in D_j(p^*)$ for $j \in M$. Such a pair is called a *Walrasian equilibrium*; p^* is a *Walrasian equilibrium price vector* (see, e.g., [5,6]). In this paper, we consider the problem of finding a Walrasian equilibrium in a multi-unit auction.

Although the Walrasian equilibrium possesses a variety of desirable properties, it does not always exist. It is known that a Walrasian equilibrium does exist in single-unit auctions under a natural assumption on bidder's valuation functions, called gross substitutes condition.

Gross Substitutes Condition and Discrete Concavity. We say that function f_j satisfies *gross substitutes (GS) condition* if it satisfies the following:

(GS) $\forall p, q \in \mathbb{R}^n_+$ with $p \leq q$, $\forall x \in D_j(p)$, $\exists y \in D_j(q)$
such that $x(i) \leq y(i)$ ($\forall i \in N$ with $p(i) = q(i)$).

This condition means that a bidder still wants to get items that do not change in price after the prices of other items increase. The concept of GS condition is introduced in Kelso and Crawford [12] for a fairly general two-sided job matching model. Since then, this condition has been widely used in various models such as matching, housing, and labor markets (see, e.g., [2,4,5,6,8,9,14]). In particular, Gul and Stacchetti [9] show the existence of a Walrasian equilibrium in a *single-unit* auction if bidders' valuation functions satisfy the GS condition; they also show that the GS condition is an "almost" necessary condition for the existence of an equilibrium in a single-unit auction.

Various characterizations of GS condition are given in the literature of discrete convex analysis and auction theory [2,8,9]. Among others, Fujishige and Yang [8] revealed the relationship between GS condition and discrete concavity called M$^\natural$-concavity. A valuation function $f_j : [\mathbf{0}, u]_\mathbb{Z} \to \mathbb{R}$ is said to be *M$^\natural$-concave* (read "M-natural-concave") if it satisfies the following:

(M$^\natural$-EXC) $\forall x, y \in [\mathbf{0}, u]_{\mathbb{Z}}$, $\forall i \in \mathrm{supp}^+(x-y)$, $\exists k \in \mathrm{supp}^+(x-y) \cup \{0\}$:
$$f_j(x) + f_j(y) \le f_j(x - \chi_i + \chi_k) + f_j(y + \chi_i - \chi_k).$$

Here, we denote $\mathrm{supp}^+(x) = \{i \in N \mid x(i) > 0\}$, $\mathrm{supp}^-(x) = \{i \in N \mid x(i) < 0\}$ for a vector $x \in \mathbb{R}^n$, $\chi_i \in \{0,1\}^n$ is the characteristic vector of $i \in N$, and $\chi_0 = \mathbf{0} = (0, 0, \dots, 0)$.

The concept of M$^\natural$-concave function is introduced by Murota and Shioura [18] as a class of discrete concave functions (independently of GS condition). It is an extension of the concept of M-concave function introduced by Murota [16]. The concepts of M$^\natural$-concavity/M-concavity play primary roles in the theory of discrete convex analysis [17].

It is shown by Fujishige and Yang [8] that GS condition and M$^\natural$-concavity are equivalent in the case of *single-unit* auctions.

Theorem 1.1. *A valuation function $f : \{0,1\}^n \to \mathbb{R}$ defined on 0-1 vectors satisfies the GS condition if and only if it is an M$^\natural$-concave function.*

This result initiated a strong interaction between discrete convex analysis and auction theory; the results obtained in discrete convex analysis are used in auction theory ([4,14], etc.), while auction theory provides discrete convex analysis with interesting applications (see, e.g., [19]).

The GS condition, however, is not sufficient for the existence of an equilibrium in a *multi-unit* setting. In the last decade, several researchers independently tried to derive conditions for valuation functions to guarantee the existence of an equilibrium in a multi-unit setting (see, e.g., [15,19]). Murota and Tamura [19] derive a stronger version of GS condition by using the relationship with M$^\natural$-concavity, and prove the existence of an equilibrium in a more general setting (see also [17, Ch. 11]). In this paper, we use the following alternative condition given in [15], which is obtained by adding to (GS) an extra inequality:

(SGS) $\forall p, q \in \mathbb{R}^n_+$ with $p \le q$, $\forall x \in D_j(p)$, $\exists y \in D_j(q)$ such that $x(i) \le y(i)$ ($\forall i \in N$ with $p(i) = q(i)$) and $\sum_{i \in N} x(i) \ge \sum_{i \in N} y(i)$.

The extra inequality $\sum_{i \in N} x(i) \ge \sum_{i \in N} y(i)$ means that if prices are increased, then a bidder wants less items than before. This condition turns out to be essentially equivalent to M$^\natural$-concavity (see Theorem 1.4 below), and also to the condition in [19]. Note that for valuation functions on $\{0,1\}^n$, the SGS condition is equivalent to the GS condition (see [15]).

Throughout this paper we assume the following conditions for all bidders' valuation functions f_j ($j = 1, \dots, m$):

(A0) f_j is monotone nondecreasing,
(A1) f_j satisfies the SGS condition,
(A2) f_j is concave-extensible,
(A3) f_j takes integer values.

The concave-extensibility (A2) is a natural condition for valuation functions [15,19]; a valuation function $f : [\mathbf{0}, u]_{\mathbb{Z}} \to \mathbb{Z}$ is said to be *concave-extensible* if there exists a concave function $\bar{f}$ defined on $\{x \in \mathbb{R}^n \mid \mathbf{0} \leq x \leq u\}$ such that $\bar{f}(x) = f(x)$ for every $x \in [\mathbf{0}, u]_{\mathbb{Z}}$. The assumption (A3) can be removed if we want to compute an ε-approximate equilibrium price vector instead of an "exact" one; for $\varepsilon > 0$, an *ε-approximate equilibrium price vector* p is defined as a vector such that $\|p - p^*\|_\infty < \varepsilon$ for some equilibrium price vector p^*. For such a problem, all results in this paper can be easily adapted with slight modifications.

Iterative Auctions and Ascending Auctions. The main theme of this paper is the computation of a Walrasian equilibrium in an ascending auction. We focus on an equilibrium price vector p^* since an allocation in the equilibrium can be computed efficiently once we obtain p^*. In the computation, we assume that bidders' valuation functions f_j are given implicitly by so-called *demand oracles*, i.e., we can get the information about demand set $D_j(p)$ for a price vector p, but no information is available about the function values of f_j. This assumption is very plausible, since bidders want to preserve their privacy about valuation functions and disclose only the information that is really needed.

In the auction literature an algorithm called the *iterative auction* (or dynamic auction, Walrasian auction, Walrasian tâtonnement process, etc.) is often used to find an equilibrium [5,6]. An iterative auction computes an equilibrium price vector by iteratively updating a current price vector p by using the information on demand sets $D_j(p)$. The most natural and popular iterative auction is the *ascending auction*, in which the current price vector is increased monotonically. The ascending auction is a natural generalization of the classical English auction for a single item, and known to have various nice properties (see, e.g., [5,6]); in particular, it is quite natural from the economic point of view, and easy to understand and implement.

In this paper, we consider the ascending auction[1] presented in Ausubel [1]. This algorithm can be seen as a simplified version of the one in Gul and Stacchetti [10], where the *Lyapunov function* defined by

$$L(p) = \textstyle\sum_{j=1}^{m} V_j(p) + u^\top p \qquad (p \in \mathbb{R}^n) \tag{3}$$

is used. It is known (see [1,21]) that p^* is an equilibrium price vector if and only if it is a minimizer of the Lyapunov function and that there exists an integral minimizer $p^* \in \mathbb{Z}^n$ of the Lyapunov function. Based on this fact, the ascending auction in [1] tries to find a minimizer of the Lyapunov function. For $X \subseteq N$, we denote by $\chi_X \in \{0,1\}^n$ the characteristic vector of X.

```
Algorithm Ascend
Step0: Set $p := p^\circ$, where $p^\circ \in \mathbb{Z}^n$ is a lower bound of some $p^* \in \arg\min L$
        (e.g., $p^\circ = \mathbf{0}$).
Step1: Find $X \subseteq N$ that minimizes $L(p + \chi_X)$.
```

[1] Our Ascend is slightly different from "Ascending Tâtonnement Algorithm" in [1] in the choice of X in Step 1; X is a minimal minimizer of $L(p + \chi_X)$ in [1], while it can be any minimizer in ours, which is easier to find than a minimal minimizer and does not increase the number of iterations. This is an additional merit of our algorithm.

Step2: If $L(p + \chi_X) = L(p)$, then output p and stop.
Step3: Set $p := p + \chi_X$ and go to Step 1.

It can be shown (cf. [1]) that this algorithm outputs an equilibrium price vector in a finite number of iterations. While the ascending auction has various nice properties (see, e.g., [5,6]), it has a disadvantage that the initial price vector must be a lower bound of some equilibrium vector. Taking this into consideration, Ausubel [1] also propose an alternative iterative auction, which allows us to start with an arbitrary price vector, but has a drawback that the change of the price vector is not monotone.

Our Contribution. The main aim of this paper is to theoretically analyze the ascending auction and other iterative auctions with respect to their time complexity. While computational experiments are often used to evaluate the practical performance of iterative auctions (see [3,20]), there is no theoretical analysis of the time complexity, even in the case of the single-unit auction, except for the termination in finite time. This paper gives the first theoretical analysis in the case of multi-unit auctions.

The results in this paper consist of the following two:

(i) Tight bounds on the number of iterations of iterative auctions,
(ii) An efficient algorithm for the update of a price vector.

Our first result is the analysis of the number of iterations required by the algorithm Ascend. The upper bound established in this paper is useful in practice by providing bidders with an a priori guarantee for the time period of the auction process. The exact bound for the number of iterations in Ascend is given in terms of the distance between the initial price vector and a minimizer of the Lyapunov function L. For the analysis, we define

$$\hat{\mu}(p) = \min\{\|p^* - p\|_\infty \mid p^* \in \arg\min L,\ p^* \geq p\} \quad (p \in \mathbb{Z}^n).$$

The value $\hat{\mu}(p)$ remains the same or decreases by one in each iteration of the algorithm. Hence, if p° is the initial vector, then $\hat{\mu}(p^\circ) + 1$ is a lower bound for the number of iterations. We show that this bound is also an upper bound.

Theorem 1.2. *Suppose that the initial vector $p^\circ \in \mathbb{Z}^n$ in the algorithm* Ascend *is a lower bound of some minimizer of the Lyapunov function L. Then, the algorithm outputs a minimizer of L and terminates exactly in $\hat{\mu}(p^\circ)+1$ iterations.*

This result shows that the trajectory of a price vector generated by Ascend is the "shortest" path between the initial vector and a minimizer of the Lyapunov function. This reveals an additional advantage of the ascending auction. We also propose some other iterative auctions in this paper and derive tight bounds for the number of iterations in these algorithms.

Our second result is concerned with the update of a price vector. The algorithm Ascend and other iterative auctions considered in this paper update the price vector by using an optimal solution of the problem $\min_{X \subseteq N} L(p + \chi_X)$

or $\min_{X \subseteq N} L(p - \chi_X)$. It is known that these problems can be reduced to submodular function minimization (SFM, for short). Although polynomial-time algorithms are available for SFM [7,17], they are quite slow and complicated.

We show that the SFM problems appearing in iterative auctions can be solved more efficiently than by a straightforward application of the existing SFM algorithms. We denote $U = \|u\|_\infty$.

Theorem 1.3. *For every integral vector $p \in \mathbb{Z}^n$, the problems $\min_{X \subseteq N} L(p + \chi_X)$ and $\min_{X \subseteq N} L(p - \chi_X)$ can be solved in $\mathrm{O}(mn^4 \log U \log(mnU))$ time.*

This improvement is achieved by exploiting the fact that valuation functions are given by demand sets and the submodular functions to be minimized can be represented in terms of demand sets as follows. For $x \in \mathbb{R}^n$ and $Y \subseteq N$, we denote $x(Y) = \sum_{i \in Y} x(i)$.

Proposition 1.1 (cf. [1,17]). *For $p \in \mathbb{Z}^n$ and $X \subseteq N$, we have*

$$L(p + \chi_X) - L(p) = -\sum_{j \in M} \min\{y(X) \mid y \in D_j(p)\} + u(X),$$

$$L(p - \chi_X) - L(p) = \sum_{j \in M} \max\{y(X) \mid y \in D_j(p)\} - u(X).$$

The problem setting of SFM in terms of demand sets is interesting in its own right.

Proofs of the results in this paper are based on the following equivalence between the SGS condition and M$^\natural$-concavity.

Theorem 1.4. *Let $f : [\mathbf{0}, u]_\mathbb{Z} \to \mathbb{Z}$ be a concave-extensible function. Then, f satisfies the SGS condition if and only if it is an M$^\natural$-concave function.*

We also point out in Corollary 2.1 that the Lyapunov function has a discrete convexity called L$^\natural$-convexity. The concepts of M$^\natural$-concavity and L$^\natural$-convexity play primary roles in the theory of discrete convex analysis [17]. On the basis of these facts, we can make full use of rich results from discrete convex analysis to prove Theorems 1.2 and 1.3.

2 Property of Indirect Utility Functions

In this section, we show that the indirect utility function $V_j : \mathbb{R}^n \to \mathbb{R}$ given by (1) is an L$^\natural$-convex function. Function $g : \mathbb{R}^n \to \mathbb{R}$ is said to be *L$^\natural$-convex* [17] if for every $p, q \in \mathbb{R}^n$ and every nonnegative $\lambda \in \mathbb{R}_+$, it holds that

$$g(p) + g(q) \geq g((p + \lambda \mathbf{1}) \wedge q) + g(p \vee (q - \lambda \mathbf{1})),$$

where $\mathbf{1} = (1, 1, \ldots, 1)$ and for $p, q \in \mathbb{R}^n$ the vectors $p \wedge q$ and $p \vee q$ denote, respectively, the vectors obtained by component-wise minimum and maximum of p and q. It is easy to see that an L$^\natural$-convex function is a submodular function on $\mathbb{R}^n$, i.e., it satisfies $g(p) + g(q) \geq g(p \wedge q) + g(p \vee q)$ $(\forall p, q \in \mathbb{R}^n)$.

Theorem 2.1. *The indirect utility function $V_j : \mathbb{R}^n \to \mathbb{R}$ is an $L^\natural$-convex function.*

Proof. The assumptions (A1) and (A2) imply the $\mathrm{M}^\natural$-concavity of valuation function f_j by Theorem 1.4. Hence, the indirect utility function V_j is $\mathrm{L}^\natural$-convex by the conjugacy theorem in discrete convex analysis [17, Ch. 8]. □

From this property we obtain the $\mathrm{L}^\natural$-convexity of the Lyapunov function L given by (3) since any linear function is also an $\mathrm{L}^\natural$-convex function and $\mathrm{L}^\natural$-convexity is closed under the addition of functions.

Corollary 2.1. *The Lyapunov function $L : \mathbb{R}^n \to \mathbb{R}$ is an $L^\natural$-convex function. In particular, L is a submodular function.*

It follows from Corollary 2.1 and the integrality assumption (A3) for valuation functions f_j that $\arg\min L$ is an integral polyhedron by the results in discrete convex analysis [17]. Since $\arg\min L$ is exactly the same as the set of equilibrium price vectors [1,21], this observation implies the known fact that there exists an integral equilibrium price vector.

3 Analysis for Number of Iterations in Iterative Auctions

In this section, we consider the algorithm Ascend and several other iterative auction algorithms for finding an integral equilibrium price vector, and analyze the number of iterations.

We first show that there exists an integral equilibrium price vector in the finite interval $[\mathbf{0}, \bar{p}]_{\mathbb{Z}}$, where $\bar{p} \in \mathbb{Z}^n_+$ is given by $\bar{p}(i) = \max_{j \in M}\{f_j(\chi_i) - f_j(\mathbf{0})\}$. Note that $\bar{p}$ can be easily computed from bidders' valuation functions.

Proposition 3.1. *There exists an equilibrium price vector p^* with $p^* \in [\mathbf{0}, \bar{p}]_{\mathbb{Z}}$.*

Hence, the number of iterations of the algorithm Ascend is at most $\sum_{i \in N} \bar{p}(i)$. We will see below that the bounds for the number of iterations in Ascend and other iterative auction algorithms are much smaller than $\sum_{i \in N} \bar{p}(i)$.

As stated in Theorem 1.2, the number of iterations in Ascend is $\hat{\mu}(p^\circ) + 1$. Its proof is quite nontrivial and can be done with the aid of some known results in discrete convex analysis. Note that any algorithm requires at least $\hat{\mu}(p^\circ) + 1$ iterations if it increases the price vector by a 0-1 vector in each iteration. Hence, the algorithm Ascend is the fastest among all iterative auction algorithms of this type, and the trajectory of the price vector is a "shortest" path from the initial vector to an equilibrium. In addition, since $\hat{\mu}(p^\circ) \leq \max_{i \in N}\{\bar{p}(i) - p^\circ(i)\}$, we can guarantee that the algorithm terminates in at most $\max_{i \in N}\{\bar{p}(i) - p^\circ(i)\} + 1$ iterations; note that this bound can be computed in advance before executing the algorithm.

Similarly to Ascend, we can consider an algorithm Descend as in [1], where the price vector is decreased by a vector $\chi_X \in \{0,1\}^n$ which is a minimizer of $L(p - \chi_X)$. It is easy to see that algorithm Descend enjoys similar properties as Ascend. We define $\check{\mu}(p) = \min\{\|p^* - p\|_\infty \mid p^* \in \arg\min L,\ p^* \leq p\}$ for $p \in \mathbb{Z}^n$.

Theorem 3.1. *Suppose that the initial vector $p^\circ \in \mathbb{Z}^n$ in the algorithm* Descend *is a upper bound of some minimizer of the Lyapunov function L. Then, the algorithm outputs a minimizer of L and terminates exactly in $\check{\mu}(p^\circ)+1$ iterations.*

An advantage of algorithms Ascend and Descend is that a price vector is updated monotonically, which is an important property from the viewpoint of auctions. They, however, have a drawback that the initial price vector should be a lower or upper bound for some minimizer of Lyapunov function L. In contrast, the following two algorithms can start from any initial price vector and find an equilibrium. Therefore, the number of iterations can be small if we can choose an initial vector that is close to some minimizer of L.

The next algorithm TwoPhase can be seen as an application of Ascend with an arbitrary initial vector, followed by Descend. The algorithm has a merit that the price vector is updated "almost" monotonically.

```
Step 0: Set $p := p^\circ$, where $p^\circ \in \mathbb{Z}^n$ is a vector with $p^\circ \in [\mathbf{0}, \bar{p}]_\mathbb{Z}$.
        Go to Ascending Phase.
Ascending Phase:
Step A1: Find $X \subseteq N$ that minimizes $L(p + \chi_X) - L(p)$.
Step A2: If $L(p + \chi_X) = L(p)$, then go to Descending Phase.
Step A3: Set $p := p + \chi_X$ and go to Step A1.
Descending Phase:
Step D1: Find $X \subseteq N$ that minimizes $L(p - \chi_X) - L(p)$.
Step D2: If $L(p - \chi_X) = L(p)$, then output $p$ and stop.
Step D3: Set $p := p - \chi_X$ and go to Step D1.
```

A version of this algorithm specialized to valuation functions defined on $\{0,1\}^n$ coincides with the one in [21]. Another algorithm called "Global Walrasian tâtonnement algorithm" in [1] repeats ascending and descending phases until a minimizer of L is found; our analysis shows that this algorithm terminates after only one ascending phase and only one descending phase.

To analyze the number of iterations required by TwoPhase, we define

$$\mu(p) = \min\{\|p^* - p\|^+_\infty + \|p^* - p\|^-_\infty \mid p^* \in \arg\min L\} \quad (p \in \mathbb{Z}^n),$$
$$\|p^* - p\|^+_\infty = \max_{i \in N} \max(0, p^*(i) - p(i)),\ \|p^* - p\|^-_\infty = \max_{i \in N} \max(0, -p^*(i) + p(i)).$$

The value $\mu(p)$ can be regarded as the "distance" between the vector p and a minimizer of L. By definition, $\mu(p)$ remains the same or decreases by one if p is updated by adding or subtracting a 0-1 vector. Hence, the algorithm TwoPhase requires at least $\mu(p^\circ) + 1$ iterations. In the following, we show that the number of iterations is bounded by $3\mu(p^\circ) + 2$.

Theorem 3.2. *The algorithm* TwoPhase *outputs an equilibrium price vector in at most $3\mu(p^\circ)+2$ iterations; more precisely, the ascending (resp., descending) phase terminates in at most $\mu(p^\circ) + 1$ iterations (resp. $2\mu(p^\circ) + 1$ iterations).*

We finally consider the algorithm Greedy, which can be seen as the steepest descent (or greedy) algorithm for the minimization of the Lyapunov function.

```
Step 0: Set $p := p^\circ$, where $p^\circ \in \mathbb{Z}^n$ is a vector with $p^\circ \in [\mathbf{0}, \bar{p}]_{\mathbb{Z}}$.
Step 1: Find $\varepsilon \in \{+1, -1\}$ and $X \subseteq N$ that minimize $L(p + \varepsilon\chi_X)$.
Step 2: If $L(p + \varepsilon\chi_X) = L(p)$, then output $p$ and stop.
Step 3: Set $p := p + \varepsilon\chi_X$ and go to Step 1.
```

This can be seen as an application of the steepest descent algorithm for L$^\natural$-convex function minimization (see [17]), for which the number of iterations is analyzed in [13]. We give a refined analysis of this algorithm in terms of the "distance" between the initial vector and a minimizer of L.

Theorem 3.3. *The algorithm* Greedy *outputs an equilibrium price vector and terminates exactly in $\mu(p^\circ) + 1$ iterations.*

As mentioned above, any iterative auction algorithm of this type requires at least $\mu(p^\circ) + 1$ iterations. Theorem 3.3 shows that Greedy is the fastest among all iterative auction algorithms of this type, and the trajectory of a price vector is a "shortest" path from the initial vector to an equilibrium. Although Greedy has such merits in the choice of the initial vector and in the number of iterations, it also has a drawback that it may repeat the increment and decrement of the price vector many times, which is not a desirable behavior from the viewpoint of auction.

It should be noted that the algorithms as well as their analysis in this section can be applied not only to the Lyapunov function but also to any general L$^\natural$-convex function since our proofs do not rely on any special structure of the Lyapunov function. In particular, the key property used in our proofs is the following property of L$^\natural$-convex functions.

Proposition 3.2 ([17, Theorem 7.7]). *Let $g : \mathbb{R}^n \to \mathbb{R}$ be an L$^\natural$-convex function. For every integral $p, q \in \mathbb{Z}^n$ with $\mathrm{supp}^+(p - q) \neq \emptyset$, it holds that $g(p) + g(q) \geq g(p - \chi_X) + g(q + \chi_X)$, where $X = \arg\max_{i \in N}\{p(i) - q(i)\}$.*

Finally, we point out that in all of the iterative auction algorithms considered in this paper we use *linear* and *anonymous* pricing rule, meaning that the price of any bundle x of goods is equal to $p^\top x$ and is the same for all bidders. In this case, we need to impose conditions on the valuation of bidders to guarantee that iterative auction algorithms work. On the other hand, so-called combinatorial auction algorithms use *nonlinear* and *discriminatory* pricing rule, i.e., the price $p(x, i)$ of a bundle x of goods depends on x and bidder i and is not linear. In this case, iterative auction algorithms work with more general valuation functions, although auction algorithms of this type are difficult to use in practice.

4 Efficient Update of Price Vector

For an update of the price vector in the ascending auction and other iterative auctions, we repeatedly solve the local optimization problems $\min_{X \subseteq N} L(p+\chi_X)$ and $\min_{X \subseteq N} L(p-\chi_X)$ for some integral $p \in \mathbb{Z}^n$, both of which can be reduced to submodular function minimization (SFM, for short). Indeed, the former problem can be reduced to the minimization of a set function given by

$$\rho_L(X) = L(p + \chi_X) - L(p) \qquad (X \subseteq N), \tag{4}$$

which is submodular since the Lyapunov function L is submodular by Corollary 2.1. The latter problem can also be reduced to SFM. In this section, we show that by using demand sets $D_j(p)$ obtained from bidders, these problems can be solved faster than a straightforward application of SFM algorithms.

In the following, we consider minimization of ρ_L given by (4). Throughout this section, we assume that for a given integral vector $p \in \mathbb{Z}^n$ and $j \in M$, a vector $x_j^\circ \in D_j(p)$ is available and the membership test in $D_j(p)$ can be done in constant time. This means that the evaluation of $\rho_L(X)$ requires solving optimization problems on $D_j(p)$, which can be done in $\mathrm{O}(mn^2 \log U)$ time, where $U = \|u\|_\infty$. Recall that SFM is solvable in polynomial time [7,17], provided the function value can be evaluated in polynomial time.

Almost all "combinatorial" polynomial-time algorithms for SFM are based on the following min-max formula (see, e.g., [7,17]). For a submodular function $\rho : 2^N \to \mathbb{Z}$, we define a set

$$B(\rho) = \{x \in \mathbb{Z}^n \mid x(Y) \leq \rho(Y) \ (\forall Y \subseteq N),\ x(N) = \rho(N)\},$$

which is called the *base polyhedron* associated with ρ.

Proposition 4.1. *For an integer-valued submodular function $\rho : 2^N \to \mathbb{Z}$,*

$$\min\{\rho(X) \mid X \subseteq N\} = \max\left\{ \textstyle\sum_{i \in N} \min\{0, x(i)\} \;\middle|\; x \in B(\rho)\right\} \tag{5}$$

holds. Moreover, if $x^* \in B(\rho)$ is an optimal solution of the maximization problem on the right-hand side of (5)*, then a set $X^* \subseteq N$ is a minimizer of ρ if and only if $\{i \in N \mid x^*(i) < 0\} \subseteq X^* \subseteq \{i \in N \mid x^*(i) \leq 0\}$ and $\rho(X^*) = x^*(X^*)$.*

Solving the maximization problem in (5) requires the membership test in $B(\rho)$. For the efficient membership test in $B(\rho)$, the existing polynomial-time algorithms use a technique to represent a vector x as a convex combination of extreme points in $B(\rho)$, which makes the algorithms slow and complicated. The fastest (weakly-)polynomial algorithm runs in $\mathrm{O}((n^4\mathrm{EO} + n^5)\log \Gamma)$ time [11], where Γ is an upper bound on $|\rho(X)|$ and EO denotes the time for function evaluation; $\Gamma = mnU$ and EO$= \mathrm{O}(mn^2 \log U)$ in our case.

We show that the minimization of ρ_L can be solved faster by using a representation of the base polyhedron $B(\rho_L)$ in terms of demand sets $D_j(p)$.

Proposition 4.2. *It holds that $B(\rho_L) = \{u - \sum_{j \in M} x_j \mid x_j \in \widetilde{D}_j(p)\ (j \in M)\}$, where $\widetilde{D}_j(p)$ is the set of minimal vectors in $D_j(p)$ for $j \in M$.*

This formula can be proven by using the representation of $L(p + \chi_X) - L(p)$ in Proposition 1.1. Note that $\widetilde{D}_j(p)$ is a base polyhedron. By Propositions 4.1 and 4.2, the minimization of ρ_L is equivalent to the problem

$$\max\left\{ \textstyle\sum_{i \in N} \min\{0, x(i)\} \;\middle|\; x = u - \sum_{j \in M} x_j,\ x_j \in \widetilde{D}_j(p)\ (j \in M)\right\}.$$

Based on this observation, we can prove Theorem 1.3. The established bound $O(mn^4 \log U \log(mnU))$ is smaller than the bound $O(mn^6 \log U \log(mnU))$ obtained by a straightforward application of the SFM algorithm in [11].

Acknowledgements. This research is supported by KAKENHI (21360045, 21740060, 24500002) and the Aihara Project, the FIRST program from JSPS.

References

1. Ausubel, L.M.: An efficient dynamic auction for heterogeneous commodities. American Economic Review 96, 602–629 (2006)
2. Ausubel, L.M., Milgrom, P.: Ascending auctions with package bidding. Front. Theor. Econ. 1, Article 1 (2002)
3. Bichler, M., Shabalin, P., Pikovsky, A.: A computational analysis of linear price iterative combinatorial auction formats. Inform. Syst. Res. 20, 33–59 (2009)
4. Bing, M., Lehmann, D., Milgrom, P.: Presentation and structure of substitutes valuations. In: Proc. EC 2004, pp. 238–239 (2004)
5. Blumrosen, L., Nisan, N.: Combinatorial auction. In: Nisan, N., et al. (eds.) Algorithmic Game Theory, pp. 267–299. Cambridge Univ. Press (2007)
6. Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial Auctions. MIT Press (2006)
7. Fujishige, S.: Submodular Functions and Optimization, 2nd edn. Elsevier (2005)
8. Fujishige, S., Yang, Z.: A note on Kelso and Crawford's gross substitutes condition. Math. Oper. Res. 28, 463–469 (2003)
9. Gul, F., Stacchetti, E.: Walrasian equilibrium with gross substitutes. J. Econ. Theory 87, 95–124 (1999)
10. Gul, F., Stacchetti, E.: The English auction with differentiated commodities. J. Economic Theory 92, 66–95 (2000)
11. Iwata, S.: A faster scaling algorithm for minimizing submodular functions. SIAM J. Comp. 32, 833–840 (2002)
12. Kelso, A.S., Crawford, V.P.: Job matching, coalition formation and gross substitutes. Econometrica 50, 1483–1504 (1982)
13. Kolmogorov, V., Shioura, A.: New algorithms for convex cost tension problem with application to computer vision. Discrete Optimization 6, 378–393 (2009)
14. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games Econom. Behav. 55, 270–296 (2006)
15. Milgrom, P., Strulovici, B.: Substitute goods, auctions, and equilibrium. J. Economic Theory 144, 212–247 (2009)
16. Murota, K.: Convexity and Steinitz's exchange property. Adv. Math. 124, 272–311 (1996)
17. Murota, K.: Discrete Convex Analysis. SIAM, Philadelphia (2003)
18. Murota, K., Shioura, A.: M-convex function on generalized polymatroid. Math. Oper. Res. 24, 95–105 (1999)
19. Murota, K., Tamura, A.: New characterizations of M-convex functions and their applications to economic equilibrium models with indivisibilities. Discrete Appl. Math. 131, 495–512 (2003)
20. Parkes, D.C., Ungar, L.H.: Iterative combinatorial auctions: theory and practice. In: Proc. 17th National Conference on Artificial Intelligence (AAAI 2000), pp. 74–81 (2000)
21. Sun, N., Yang, Z.: A double-track adjustment process for discrete markets with substitutes and complements. Econometrica 77, 933–952 (2009)

New Results on the Online Pricing Problem

Xiangzhong Xiang

Department of Computer Science
The University of Hong Kong
xzxiang@cs.hku.hk

Abstract. We study the online pricing problem where there are m identical items on sale and a sequence of users $u_1, u_2, \ldots$ arrive one by one. Each u_i has a non-increasing acceptable price function $\varphi_i(\cdot)$ where $\varphi_i(x)$ is the highest unit price u_i is willing to pay for x items. Upon the arrival of a user, the seller needs to decide the number of items to be sold to the user and at what price. The goal is to maximize the revenue of the seller, which is the sum of money paid by all users.

For the case when the items are divisible and can be sold fractionally, we improve the results of Zhang *et al.* [19]. We design a new deterministic algorithm $\mathcal{D}p$-DIV and prove that it has competitive ratio $O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$ where h is the highest unit price a user is willing to pay; our algorithm is substantially better than the $O(h^{3\log_2^{-1/2} h})$ - competitive algorithm of Zhang *et al.* We also prove that no randomized algorithm can do better than $\Omega(\log h)$-competitive.

For the case when items are indivisible, there is no known result. We show in this paper that any deterministic algorithm for this case must have competitive ratio at least $\Omega(h)$-competitive. Then, we give the first competitive randomized algorithm $\mathcal{R}p$-INDIV with competitive ratio $O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$. If h is known ahead of time, we can reduce the competitive ratio to $O(\log h)$. Besides, we prove that no randomized algorithm can do better than $\Omega(\log h)$-competitive.

1 Introduction

The pricing problem, as well as its many variants, have been studied extensively in recent years and have found important applications in economics [1, 4, 10, 12, 14, 15]. This paper studies an online version of the problem, in which there is a sequence of users who want to buy goods from one seller. These users are arriving one by one at different times, and each user would specify how much she is willing to pay for a bundle of goods. Upon the arrival of a user, the seller needs to determine immediately the number of goods to be sold to the user and the price he would charge her. His goal is to maximize the sum of money paid by users. An example of such setting is the bandwidth allocation on a communication link where the total demand for bandwidth exceeds the link's capacity. Requests for bandwidth arrive at different times and each request needs an immediate answer about the allocation and the price of bandwidth. Similar applications can be found in the allocation of valuable resources such as CPU time or cache space.

The online pricing problem was first studied by Zhang *et al.* [17]. They focused on a special version of the problem, namely the *online pricing for divisible goods* problem, in which all goods are divisible and can be sold fractionally. They gave a deterministic algorithm for the problem with competitive ratio $O(h^{3\log_2^{-1/2} h})$ where h is the highest unit price (i.e., the price for one unit of item) that at least some user is willing to pay. Note that in general, h is only known when all users have arrived. However, if h is known before any user arrives, they showed that their algorithm can be improved to $O(\log h)$-competitive. In [19], Zhang *et al.* proved that any deterministic algorithm for the online pricing for divisible goods problem must have competitive ratio at least $\frac{1}{2}\lfloor \log_2 h \rfloor$.

We note that there is yet no known result for the other version of the problem, namely the *online pricing for indivisible goods* problem, in which all goods are indivisible.

Our Results. In this paper, we tackle both versions of the online pricing problems. For the online pricing for indivisible goods problem, we show that any deterministic algorithm for the problem must have competitive ratio $\Omega(h)$. Then, we give the first competitive randomized algorithm $\mathcal{R}p$-INDIV for this problem. The algorithm is surprisingly simple and uses very little power of randomization; when it starts, it picks randomly a unit price p according to some fixed probability distribution, and whenever a new user arrives, it simply sells her the largest number of goods that she is willing to buy under this unit price p. The difficult part is to choose the right probability distribution. Also, the competitive analysis is not easy. By using a good probability distribution, we prove that $\mathcal{R}p$-INDIV has a competitive ratio $O((\log h)^{1+\epsilon})$ where $\epsilon > 0$ is a very small number. By using a more elaborate probability distribution, we achieves a competitive ratio of $O(\log h \log^{(2)} h \ldots \log^{(\Delta)} h)$ where $\Delta = \log^* h - 1$. Here, $\log^{(1)} h = \log h$, $\log^{(j)} h = \log(\log^{(j-1)} h)$ for $j \geq 2$, and $\log^* h$ is the iterated logarithm of h, which is the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1. Furthermore, we show that if h is known ahead of time, we can choose an even better distribution, which would reduce the competitive ratio of $\mathcal{R}p$-INDIV to $O(\log h)$. Finally, we prove that any randomized algorithm for the problem must have competitive ratio $\Omega(\log h)$.

For the online pricing for divisible goods problem, our major contribution is to improve the result of Zhang *et al.* [17,19]. We give a new deterministic algorithm $\mathcal{D}p$-DIV, which uses a more refined pricing structure and a more elaborate selling strategy than those proposed in [17, 19]. Our strategy is inspired by our randomized algorithm $\mathcal{R}p$-INDIV and we prove that $\mathcal{D}p$-DIV has competitive ratio $O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$, which is a significant improvement of Zhang *et al.*'s $O(h^{3\log_2^{-1/2} h})$-competitive algorithm. Recall that every deterministic algorithm for the online pricing for divisible goods problem must have competitive ratio at least $\frac{1}{2}\lfloor \log h \rfloor$ [19]. We show in this paper that we cannot do much better even if randomization is allowed; we prove that any randomized algorithm must have competitive ratio $\Omega(\log h)$. We note that our randomized algorithm $\mathcal{R}p$-INDIV for

Table 1. Summary of results on the competitive ratios (all of our lower bound results hold even for the case when h is known before any user arrives)

	Our results
deter-ministic	$\Omega(h)$
random	$\Omega(\log h)$
	$O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$ $O(\log h)$ if h known

Pricing for indivisible goods (No previous result)

	Previous results	Our results
deter-ministic	$\Omega(\log h)$	-
	$O\left(h^{\frac{3}{\sqrt{\log_2 h}}}\right)$	$O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$
random	-	$\Omega(\log h)$
	-	$O(\log h)$ if h known

Pricing for divisible goods

indivisible goods can also be applied directly to solve this problem for divisible goods, and we can prove that this gives us a competitive ratio $O(\log h)$ when h is known. Table 1 summarizes all results.

Related Works. The offline pricing problem was firstly introduced by Guruswami *et al.* [12]. Guruswami *et al.* gave an approximation algorithm for the pricing problem with unlimited supplies of m different types of items and n *single-minded* users (i.e., each user is only interested in a particular set of items); their algorithm achieves expected revenue within an $O(\log n + \log m)$ factor of the total social welfare. This approximation ratio was proved to be tight by Briest [5] and Chalermsook *et al.* [7]. For the case when each single-minded bidder wants at most k types of items, Briest *et al.* [6] gave an $O(k^2)$-approximation algorithm, and later Belcan *et al.* [2] improved the approximation ratio to $O(k)$. For the case when the m types of items on sale have limited supply and users are *unit-demanded* (i.e., each user would like to buy at most one item), Guruswami *et al.* [12] gave an $O(\log m)$-approximation algorithm.

In offline settings, the pricing problem can be viewed as combinatorial auctions, which have been studied extensively [4, 8, 10, 11, 13]. One direction studied in combinatorial auctions is envy-freeness, which means that given the pricing, no user would prefer to be assigned a different bundle of items [8,11,13]. Another direction about combinatorial auctions is incentive compatibility, where rational bidders are always motivated to bid their true valuations [4,10,14].

For online pricing, Zhang *et al.* [18] have also considered a variant of the online pricing problem called online pricing for multi-type of items, in which the seller has k types of items, each type has m copies, and users arrive online. Each user is single-minded. The seller must decide the unit price and the number of bundles sold to a user when she arrives. Bundles can be sold fractionally. Zhang *et al.* showed that any deterministic algorithm for this problem must have competitive ratio at least $\Omega(\log h + \log k)$, where h is the highest unit price, and under the assumption that h is known ahead of time, they gave an $O(\sqrt{k} \cdot \log h \log k)$-competitive deterministic algorithm for the problem.

Balcan *et al.* [3] also considered the problem of pricing n types of items to maximize revenue when faced with a series of unknown users arriving online. Compared with the models in [17–19], Balcan *et al.*'s model assumes that for each type of item, the user only demands one copy instead of many copies. In the unlimited supply setting, they showed that a random single price achieves expected revenue within a logarithmic factor of the total social welfare. In the limited supply setting, they showed that a random single price achieves revenue within a factor of $2^{O(\sqrt{\log n \log \log n})}$ of the total social welfare. Cole *et al.* [9] considered an online problem where one of m identical items is offered for sale at each time unit, users arrive and depart dynamically and each user is interested in winning one item; the goal is to maximize the sum of the utilities of winning bidders and they gave a constant-competitive algorithm.

2 Problem Definitions

The online pricing for indivisible goods problem is defined formally by Zhang *et al.* [17] as follows: There is a seller with m identical items, and a sequence of users $u_1, u_2, \ldots$ come one by one. When a user arrives, the seller must determine the number of items to be sold to the user, as well as the unit price for selling these items. Each item is indivisible and can be sold to only one user. Every user u_i has a non-increasing acceptable price function $\varphi_i(\cdot)$ where $\varphi_i(x)$ is the highest unit price u_i is willing to pay for buying x items [1] . Let $h = \max_{i,x} \varphi_i(x)$ be the highest value among all $\varphi_i(x)$. Suppose that a user u_i arrives, and that the seller decides that he can sell m_i items to u_i with a unit price of p_i. If $p_i > \varphi_i(m_i)$, user u_i cannot accept the price and will not buy any item; otherwise, user u_i will accept the price, pay $m_i \cdot p_i$ to the seller and get m_i items. The objective is to maximize the revenue of the seller, which is the sum of money paid by all users. The online pricing for divisible goods problem is defined very similarly; the only difference is that each item is divisible and can be sold fractionally.

Let A be a deterministic algorithm for the online pricing problem (either for indivisible or divisible goods). Given m items and a user sequence σ, let $ALG(m, \sigma)$ and $OPT(m, \sigma)$ denote the total revenue of A and that of an optimal offline algorithm, respectively. The competitive ratio of A is defined to be $c_A = \sup_{m,\sigma} \frac{OPT(m,\sigma)}{ALG(m,\sigma)}$. Now suppose that A is a randomized algorithm. Let $ALG(m, \sigma, \delta)$ denote the total revenue of A based on the random choice δ. Let the expected revenue be $E_\delta(ALG(m, \sigma, \delta))$. Then the competitive ratio of A is defined to be $c_A = \sup_{m,\sigma} \frac{OPT(m,\sigma)}{E_\delta(ALG(m,\sigma,\delta))}$.

The rest of the paper is structured as follows: In Section 3, we discuss the online pricing for indivisible goods and present the competitive randomized algorithm $\mathcal{R}p$-INDIV. In Section 4, we discuss the online pricing for divisible goods and give the competitive deterministic algorithm $\mathcal{D}p$-DIV.

[1] To simplify the analysis, we assume without loss of generality that the seller would only sell items at unit price no less than 1.

3 Online Pricing for Indivisible Goods

In this section, we consider online pricing for indivisible goods. We prove that no deterministic algorithm can do better than h-competitive. A randomized algorithm $\mathcal{R}p$-INDIV is presented and it is proved to be $O(\prod_{j=1}^{\log^* h-1} \log^{(j)} h)$-competitive when h is unknown. If h is known ahead of time, the competitive ratio can be decreased to $2\lfloor \log_2 h \rfloor + 2$. We also prove that the competitive ratio of any randomized algorithm is at least $\frac{\lceil \log_2(h+1) \rceil + 1}{2}$.

We show a lower bound for deterministic algorithms first:

Theorem 1. *No deterministic algorithm for the online pricing for indivisible goods problem can have a competitive ratio less than h.*

Proof. Suppose that there is only $m = 1$ item on sale. First, user u_1 arrives and is willing to pay 1 for each item. Consider two cases: (1) If u_1 is assigned the item by the seller, we send another user u_2 who is willing to pay h for each item, and stop. The revenue returned by the online algorithm is 1, while the optimal revenue is h by assigning the item to u_2 at a unit price of h. (2) If u_1 is not assigned the item, then we stop; in such case, the revenue returned by the online algorithm is 0, while the optimal revenue is 1. In both cases, the ratios between the optimal revenue and the revenue of the algorithm are no less than h.

3.1 A Competitive Randomized Algorithm: $\mathcal{R}p$-INDIV

$\mathcal{R}p$-INDIV is shown below. We choose a non-negative integer κ randomly and set $\tau = 2^\kappa$ as the unit price for all users. We will specify the probability distribution $\Pr[\kappa = i] = q_i$ later. When a new user arrives, we try to allocate her the largest number that she is willing to buy given the unit price τ.

```
1 Choose an integer $\kappa$ randomly from the set $N = \{0, 1, 2, \dots\}$ with probability
  $\Pr[\kappa = i] = q_i$;
2 Set $\tau = 2^\kappa$;
3 $x := m$; // $x$ is the number of the remaining available items
4 while a new user $u_j$ arrives do
5     Let $y$ be the largest number that the user is willing to buy given the unit
      price $\tau$, i.e., the largest $y$ with $\varphi_j(y) \geq \tau$;
6     Set unit price to be $p_j = \tau$;
7     Assign $m_j = \min\{y, x\}$ items to the user;
8     $x := x - m_j$;
9 end
```

Algorithm 1. $\mathcal{R}p$-INDIV

Note that in the algorithm, all the variables x, y, m_j will be integers. We now prove some useful properties on $\mathcal{R}p$-INDIV before analyzing its competitive ratio. Let ALG be the random variable which denotes the the total revenue returned by $\mathcal{R}p$-INDIV. For any $i \geq 0$, let $ALG_{\geq 2^i}$ be the revenue returned by $\mathcal{R}p$-INDIV if the unit price τ is 2^i, or equivalently, when $\kappa = i$. We are interested in the expected value of ALG, which is defined to be

$$E(ALG) = \sum_{0 \le i \le \infty} ALG_{\ge 2^i} \Pr(\kappa = i) = \sum_{0 \le i \le \infty} ALG_{\ge 2^i} \, q_i. \qquad (1)$$

Assume that each bidder j is assigned m'_j items at unit price p'_j in the optimal offline algorithm. Let O be the set of users assigned $m'_j > 0$ items in the optimal offline algorithm. Define $O_{[2^i,2^{i+1})} = \{j \in O \mid p'_j \in [2^i, 2^{i+1})\}$ to be the users in O whose unit prices are in the interval $[2^i, 2^{i+1})$. Use $OPT_{[2^i,2^{i+1})}$ to denote the revenue from users in $O_{[2^i,2^{i+1})}$ in optimal offline algorithm.

Lemma 1. *For any $i \ge 0$, $ALG_{\ge 2^i} \ge \frac{1}{2} OPT_{[2^i,2^{i+1})}$.*

Proof. In $\mathcal{R}p$-INDIV, the unit price for all users is $\tau = 2^i$. Assume each user j is willing to buy at most y_j items at unit price 2^i. When new user j arrives, $\mathcal{R}p$-INDIV will try to assign y_j items to j until all the m items are sold out. So:

$$ALG_{\ge 2^i} = \min\{\sum_j y_j, m\} \, 2^i \ge \min\{\sum_{j \in O_{[2^i,2^{i+1})}} y_j, m\} \, 2^i.$$

For user $j \in O_{[2^i,2^{i+1})}$, as $\varphi_j(\cdot)$ is non-increasing, we have $m'_j \le y_j$. Thus:

$$\begin{aligned} ALG_{\ge 2^i} \ge \min\{\sum_{j \in O_{[2^i,2^{i+1})}} m'_j, m\} \, 2^i &= \sum_{j \in O_{[2^i,2^{i+1})}} m'_j \, 2^i \\ &\ge \frac{1}{2} \sum_{j \in O_{[2^i,2^{i+1})}} m'_j \, p'_j = \frac{1}{2} OPT_{[2^i,2^{i+1})}. \end{aligned}$$

We are now ready to describe how to choose the random unit price τ to make $\mathcal{R}p$-INDIV competitive. Consider the case when h is unknown first. Consider any fixed number $\epsilon > 0$. Let $\alpha = \sum_{0 \le i \le \infty} \frac{1}{(1+i)^{1+\epsilon}}$. The following lemma will give the competitive ratio of $\mathcal{R}p$-INDIV under the probability distribution $q_i = \frac{1}{\alpha(1+i)^{1+\epsilon}}$. (Note that $\sum_{i \ge 0} q_i$ is just equal to 1.)

Lemma 2. *If h is unknown, by setting $q_i = \frac{1}{\alpha(i+1)^{1+\epsilon}}$, $\mathcal{R}p$-INDIV has competitive ratio $O((\log_2 h)^{1+\epsilon})$.*

Proof. Since h is the highest price, $ALG_{\ge 2^i}$ and $OPT_{[2^i,2^{i+1})}$ are 0 for all $i \ge \lfloor \log_2 h \rfloor + 1$. Then by Equation (1) and Lemma 1,

$$\begin{aligned} E(ALG) = \sum_{0 \le i \le \lfloor \log_2 h \rfloor} ALG_{\ge 2^i} \, q_i \ge \\ \frac{1}{2} \sum_{0 \le i \le \lfloor \log_2 h \rfloor} OPT_{[2^i,2^{i+1})} \frac{1}{\alpha(i+1)^{1+\epsilon}} \ge \frac{1}{2\alpha(\lfloor \log_2 h \rfloor + 1)^{1+\epsilon}} OPT \end{aligned}$$

, the theorem follows.

Consider a more elaborate probability distribution now: $q_i = \frac{1}{\alpha' f(i+1)}$ where $f(x) = x \cdot \prod_{j=1}^{\log_2^*(x)-1} \log_2^{(j)} x$ and $\alpha' = \sum_{0 \le i \le \infty} \frac{1}{f(i+1)}$. Note that the sum $\sum_{0 \le i \le \infty} \frac{1}{f(i+1)}$ converges as the integral $\int_1^\infty \frac{dx}{f(x)}$ converges. [2] Then we have the following theorem:

[2] Details can be found in the full version of this paper (`http://www.cs.hku.hk/~xzxiang/papers/OnlinePricing.pdf`).

Theorem 2. *If h is unknown, by setting* $q_i = \frac{1}{\alpha' \cdot (i+1) \prod_{j=1}^{\log^*(i+1)-1} \log^{(j)}(i+1)}$, $\mathcal{R}p$-INDIV *has competitive ratio* $O(\prod_{j=1}^{\log^*(h)-1} \log^{(j)} h)$.

Proof. Similar to the proof of Lemma 2.

The following theorem shows that if h is known ahead of time, then selecting a random κ uniformly will make $\mathcal{R}p$-INDIV more competitive.

Theorem 3. *If h is known ahead of time, let $g = \lfloor \log_2 h \rfloor + 1$. By choosing $\kappa \in \{0, 1, \ldots, g-1\}$ uniformly, i.e., set $q_i = 1/g$ for every $0 \le i \le g-1$, and $q_i = 0$ for all other i, $\mathcal{R}p$-INDIV achieves a competitive ratio of $2\lfloor \log_2 h \rfloor + 2$.*

Proof. Since $g = \lfloor \log_2 h \rfloor + 1$ and the highest price is h, we have that $OPT = \sum_{0 \le i \le g-1} OPT_{[2^i, 2^{i+1})}$. Then, by Equation (1) and Lemma 1,

$$E(ALG) = \textstyle\sum_{0 \le i \le g-1} ALG_{\ge 2^i} q_i \ge \frac{1}{2} \sum_{0 \le i \le g-1} OPT_{[2^i, 2^{i+1})} \frac{1}{g} = \frac{1}{2g} OPT$$

, the theorem follows.

3.2 Lower Bound for Randomized Algorithms

Consider any randomized algorithm A for the online pricing for indivisible goods problem. Now we derive a lower bound on its competitive ratio, using a technique inspired by Yao's lemma [16].

First, we construct the input as follows: Let $g = \lceil \log_2(h+1) \rceil$. There are m indivisible items on sale and g users, $u_0, u_1, \ldots, u_{g-1}$. For each $0 \le i \le g-1$, the user u_i is willing to pay 2^i for each item.

Now, we consider the following input distribution $\mathcal{I} = \{I_0, I_1, \ldots, I_{g-1}\}$ where I_i is the input in which the adversary stops giving any user after the sequence of users $u_0, u_1, \ldots, u_i$ have arrived. The probability of having I_i is:

- For any $0 \le i \le g-2$, the input is I_i with probability $q_i = \frac{1}{2^{i+1}}$.
- The input is I_{g-1} with probability $q_{g-1} = \frac{1}{2^{g-1}}$.

For any $I_i \in \mathcal{I}$, let OPT_{I_i} be the revenue returned by the offline optimal algorithm on input I_i, and $ALG_{I_i,\delta}$ is the one returned by A on input I_i and random choice δ.

Theorem 4. *Let c be the competitive ratio of any randomized algorithm A, then $c \ge \frac{\lceil \log_2(h+1) \rceil + 1}{2}$.*

Proof. Since $OPT_{I_i} \le c \cdot E_\delta[ALG_{I_i,\delta}]$ for any $I_i \in \mathcal{I}$,

$$E_{I_i}[OPT_{I_i}] \le E_{I_i}[c \cdot E_\delta[ALG_{I_i,\delta}]] \le c \cdot E_\delta[E_{I_i}[ALG_{I_i,\delta}]] \le c \cdot \max_\delta E_{I_i}[ALG_{I_i,\delta}]. \tag{2}$$

Note that

$$E_{I_i}[OPT_{I_i}] = \textstyle\sum_{0 \le i \le g-1} q_i \cdot m \cdot 2^i = \sum_{0 \le i \le g-2} \frac{1}{2^{i+1}} \cdot m \cdot 2^i + \frac{1}{2^{g-1}} \cdot m \cdot 2^{g-1} = \frac{(g+1) \cdot m}{2}. \tag{3}$$

Below, we prove that for any fixed δ, $E_{I_i}[ALG_{I_i,\delta}] \leq m$. Then by (2) and (3), we conclude that $c \geq \frac{g+1}{2}$ and the theorem follows.

Consider any deterministic algorithm D. Let D_{I_i} be the revenue returned by D on input I_i. Suppose that I_{g-1} is given to D as input. Assume that in D, m_j items are assigned to user u_j at unit price no more than 2^j for each $0 \leq j \leq g-1$. By the deterministic nature of D, we conclude that when the input is I_i, for $0 \leq j \leq i$, user u_j is still assigned m_j items at the same unit price as before. Thus

$$\begin{aligned} E_{I_i}[D_{I_i}] \leq \textstyle\sum_{0\leq i\leq g-1} q_i \cdot \sum_{0\leq j\leq i} 2^j m_j &= \textstyle\sum_{0\leq j\leq g-1} 2^j m_j \sum_{j\leq i\leq g-1} q_i \\ &= \textstyle\sum_{0\leq j\leq g-1} 2^j m_j \frac{1}{2^j} = \sum_{0\leq j\leq g-1} m_j \leq m. \end{aligned}$$

Therefore, we conclude, $E_{I_i}(D_{I_i}) \leq m$. Note that when δ is fixed, A is essentially a deterministic algorithm and therefore $E_{I_i}[ALG_{I_i,\delta}] \leq m$, as required.

4 Online Pricing for Divisible Goods

In this section, we consider online pricing for divisible goods. We present a deterministic algorithm $\mathcal{D}p$-DIV which is $O(\prod_{j=1}^{\log^*(h)-1} \log^{(j)} h)$-competitive when h is unknown. If h is known, the competitive ratio can be decreased to $4\lfloor \log_2 h \rfloor + 6$. We also show that no randomized algorithm can do better than $\frac{\lceil \log_2(h+1)\rceil+1}{2}$-competitive. The randomized algorithm $\mathcal{R}p$-INDIV can be applied to solve this problem and achieves the same asymptotic competitive ratios as $\mathcal{D}p$-DIV. Because of the page limits, we move results about randomized algorithms into the full version of this paper (`http://www.cs.hku.hk/~xzxiang/papers/OnlinePricing.pdf`).

4.1 A Deterministic Algorithm: $\mathcal{D}p$-DIV

Recall that users are arriving online. Consider a scenario that the first user u_1 is willing to pay a unit price of 1 for any number of items. If the seller assigns all items to u_1, the second user u_2 may arrive and is willing to pay a unit price of h; if the seller assigns no item to u_1, maybe no user comes anymore and the revenue is 0. Both strategies have poor performance. This example shows that the seller should bound the number of items when selling them at low prices.

Our algorithm $\mathcal{D}p$-DIV is shown in Algorithm 2. We will specify the parameters q_i's later. In $\mathcal{D}p$-DIV, we set the unit price for each user to 2 to the power of a non-negative integer, i.e., 2^i ($i \in \{0, 1, \dots\}$). Associate $m \cdot q_i$ items to the price level 2^i (note that $\sum_{i\geq 0} q_i = 1$). For items in price level 2^i, they must be sold at unit price no less than 2^i. If all items in price level 2^i are assigned to some users, the seller may use the remaining quota from lower price levels to satisfy the user with unit price 2^i. When using the remaining quota from lower price levels, the order must be strictly decreasing, i.e., first try price level 2^{i-1}, then $2^{i-2}, \dots$ In $\mathcal{D}p$-DIV, we use x_i to denote the current maximum number of items which can be used by users with unit price 2^i. Note that in the initial step, $x_i = \sum_{0\leq j\leq i} mq_i$.

For example, the current maximum numbers of items available for price levels are: $x_0 = 1, x_1 = 3, x_2 = 5$ and $x_3 = 10$. Now we sell 3 items to some user at unit price 2^2. Note that there are $x_2 - x_1 = 2$ unused items in level 2^2 and $x_1 - x_0 = 2$ unused items in level 2^1. Then 2 items in level 2^2 and 1 item in level 2^1 will be used. The updated numbers will be: $x_0 = 1, x_1 = 2, x_2 = 2$ and $x_3 = 7$.

```
1  Initialize parameters $q_0, q_1, \ldots$ such that $\sum_{i\geq 0} q_i = 1$ and $q_0 \geq q_1 \geq \ldots$ ;
2  Associate $m \cdot q_i$ items to price layer $2^i$ $(i = 0, 1, \ldots)$;
3  Set $x_i := \sum_{0\leq j\leq i} m \cdot q_j$ $(i = 0, 1, \ldots)$;
4  while a new user arrives do
5      For each $j \geq 0$, let $y_j$ be the largest number that the user is willing to buy
       given price $2^j$ and satisfying $y_j \leq m$;
6      Let $t$ be the minimum value such that $x_t > 0$. If there exists no such $t$, exit;
7      Let $\kappa = \arg\max_{j\geq t} y_j \cdot 2^j$;
8      Set unit price to be $p_i = 2^\kappa$;
9      Assign $m_i = \min\{x_\kappa, y_\kappa\}$ items to the user at unit price $p_i$;
       // Modify Available Numbers of Items
10     if $m_i == x_\kappa$ then
11         $x_j := 0$ for $0 \leq j \leq \kappa$;
12     else
13         Let $\ell$ be the maximum value such that $x_\kappa - x_\ell \geq y_\kappa$;
14         for $j = \ell + 1$ to $\kappa$ do
15             $x_j := x_\kappa - y_\kappa$;
16         end
17     end
18     for $j > \kappa$ do
19         $x_j := x_j - m_i$;
20     end
21 end
```

Algorithm 2. $\mathcal{D}p$-DIV

We begin to analyze the performance of $\mathcal{D}p$-DIV. We say the price level 2^i is full if $x_i = 0$, in other words, all items in price levels from 2^0 to 2^i are used up. After running $\mathcal{D}p$-DIV, if all x_j are 0, set $k = \lfloor \log_2 h \rfloor$; otherwise, let k be the highest value such that $x_k = 0$, i.e., all price levels from 2^0 to 2^k are full while price level 2^{k+1} is not full. As the highest price is h, $k \leq \lfloor \log_2 h \rfloor$ is always true.

Use ALG to denote the revenue returned by $\mathcal{D}p$-DIV, OPT to denote that returned by the optimal offline algorithm. Partition OPT into two parts: $OPT1$ and $OPT2$. $OPT1$ is the revenue for users with assigned price less than 2^{k+1} by the optimal offline algorithm. $OPT2$ is the revenue for all the other users.

Lemma 3. $OPT1 \leq \frac{2}{q_k} \cdot ALG$

Proof. $OPT1$ is less than $m \cdot 2^{k+1}$ as the unit prices of each items are less than 2^{k+1} and there are at most m items. In $\mathcal{D}p$-DIV, the unit price for any item in price level 2^i is at least 2^i. Since the price levels from 2^0 to 2^k are full,

$$ALG \geq \sum_{i=0}^{k} (m \cdot q_i) \cdot 2^i \geq m \cdot q_k \cdot 2^k = \frac{q_k}{2} \cdot (m \cdot 2^{k+1}) \geq \frac{q_k}{2} \cdot OPT1$$

Lemma 4. *$OPT2 \leq (2 + 2/q_k) \cdot ALG$*

Proof. Recall that $OPT2$ is the revenue for all the users with assigned price no less than 2^{k+1} by the optimal offline algorithm. We consider two cases:

- User u_i is assigned with unit price $p' \geq 2^{k+1}$ by the optimal offline algorithm and with unit price $2^p > 2^k$ by $\mathcal{D}p$-DIV.
  Assume that in the optimal offline algorithm, u_i is assigned m' items. Let $2^o \leq p' < 2^{o+1}$. y_o and y_p are the largest number of items u_i would like to buy at unit price 2^o and 2^p. As $p' \geq 2^o$ and $\varphi_i(\cdot)$ is non-increasing, $m' \leq y_o$. The revenue in optimal algorithm is: $m' \cdot p' \leq y_o \cdot 2^{o+1} = 2 \cdot y_o \cdot 2^o$. Recall that in Line 7 of $\mathcal{D}p$-DIV, we choose the price level 2^j which maximizes $y_j \cdot 2^j$; as price level 2^p is chosen now, we get that $y_p \cdot 2^p \geq y_o \cdot 2^o$.
  Since the price level 2^p is not full at the end, u_i is assigned y_p items at unit price 2^p by $\mathcal{D}p$-DIV. His revenue is at least half of that in optimal algorithm as $y_p \cdot 2^p \geq y_o \cdot 2^o \geq \frac{1}{2} \cdot m' \cdot p'$.
- User u_i is assigned with unit price $p' \geq 2^{k+1}$ by the optimal offline algorithm and with unit price $2^p \leq 2^k$ by $\mathcal{D}p$-DIV.
  Similarly, we assume that in optimal algorithm, u_i is assigned m' items at unit price $2^o \leq p' < 2^{o+1}$. y_o and y_p are the largest number of items u_i would like to buy at unit price 2^o and 2^p. We still have: $y_p \cdot 2^p \geq y_o \cdot 2^o \geq \frac{1}{2} \cdot m' \cdot p'$. If all the y_p items are assigned to u_i, his revenue is at least half of that in optimal algorithm. Otherwise, $\mathcal{D}p$-DIV assigns $x_p < y_p$ items to u_i (as shown in Line 9 of $\mathcal{D}p$-DIV). After that, the price level 2^p is full. The total revenue of all items in that level 2^p is at least:
  $$(m \cdot q_p) \cdot 2^p \geq q_p \cdot y_p \cdot 2^p \geq \frac{1}{2} \cdot q_p \cdot m' \cdot p' \geq \frac{q_k}{2} \cdot (m' \cdot p') \tag{4}$$
  The last inequality is true as $q_0 \geq \cdots \geq q_k$. For each level $2^p \leq 2^k$, there is at most one user with assigned number $x_p < y_p$ by $\mathcal{D}p$-DIV.

We partition $OPT2$ into two parts further: $OPT21$ and $OPT22$. $OPT21$ is the revenue of users who are assigned y_p items by $\mathcal{D}p$-DIV while $OPT22$ is that of users who are assigned $x_p < y_p$ items by $\mathcal{D}p$-DIV. We have shown that $ALG \geq \frac{1}{2} \cdot OPT21$. As for $OPT22$, there is at most one user with assigned number $x_p < y_p$ in each level by $\mathcal{D}p$-DIV. By (4), we get: $ALG \geq \frac{q_k}{2} \cdot OPT22$. Thus, $OPT2 = OPT21 + OPT22 \leq (2 + 2/q_k) \cdot ALG$

Lemma 5. *$OPT \leq (4/q_{\lfloor \log_2 h \rfloor} + 2) \cdot ALG$.*

Proof. By Lemma 3 and 4, we have $OPT = OPT1 + OPT2 \leq (4/q_k + 2) \cdot ALG$. As $k \leq \lfloor \log_2 h \rfloor$, the lemma follows.

We are now ready to specify how to choose the parameter q_i to make $\mathcal{D}p$-DIV competitive. Consider the case when h is unknown. Consider any fixed number $\epsilon > 0$. Let $\alpha = \sum_{0 \leq i \leq \infty} \frac{1}{(1+i)^{1+\epsilon}}$. The following theorem gives the competitive ratio of $\mathcal{D}p$-DIV under the setting $q_i = \frac{1}{\alpha(1+i)^{1+\epsilon}}$. (Note that $\sum_{i \geq 0} q_i = 1$.)

Lemma 6. *If h is unknown, by setting $q_i = \frac{1}{\alpha(i+1)^{1+\epsilon}}$, $\mathcal{D}p$-DIV has competitive ratio $O((\log_2 h)^{1+\epsilon})$.*

Proof. By Lemma 5, $OPT/ALG \leq 4/q_{\lfloor \log_2 h \rfloor} + 2 \leq 4\alpha(\lfloor \log_2 h \rfloor + 1)^{1+\epsilon} + 2$.

Consider the setting $q_i = \frac{1}{\alpha' f(i+1)}$ where $f(x) = x \cdot \prod_{j=1}^{\log_2^*(x)-1} \log_2^{(j)} x$ and $\alpha' = \sum_{0 \leq i \leq \infty} \frac{1}{f(i+1)}$. Then we have the following theorem:

Theorem 5. *If h is unknown, by setting $q_i = \frac{1}{\alpha'(i+1)\prod_{j=1}^{\log^*(i+1)-1} \log^{(j)}(i+1)}$, $\mathcal{R}p$-INDIV has competitive ratio $O(\prod_{j=1}^{\log^*(h)-1} \log^{(j)} h)$.*

Proof. Similar to the proof of Lemma 6.

The following theorem considers the case when h is known ahead of time.

Theorem 6. *If h is known ahead of time, by setting $q_i = \frac{1}{\lfloor \log_2 h \rfloor + 1}$ for every $0 \leq i \leq \lfloor \log_2 h \rfloor$, and $q_i = 0$ for all other i, $\mathcal{D}p$-DIV achieves a competitive ratio of $4\lfloor \log_2 h \rfloor + 6$.*

Proof. By Lemma 5, $OPT/ALG \leq 4/q_{\lfloor \log_2 h \rfloor} + 2 \leq 4\lfloor \log_2 h \rfloor + 6$.

References

1. Abraham, I., Babaioff, M., Dughmi, S., Roughgarden, T.: Combinatorial auctions with restricted complements. In: Proceedings of the 13th ACM Conference on Electronic Commerce, pp. 3–16. ACM (2012)
2. Balcan, M.-F., Blum, A.: Approximation algorithms and online mechanisms for item pricing. In: Proceedings of the 7th ACM Conference on Electronic Commerce, pp. 29–35. ACM (2006)
3. Balcan, M.-F., Blum, A., Mansour, Y.: Item pricing for revenue maximization. In: Proceedings of the 9th ACM Conference on Electronic Commerce, pp. 50–59. ACM (2008)
4. Bartal, Y., Gonen, R., Nisan, N.: Incentive compatible multi unit combinatorial auctions. In: Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge, pp. 72–87. ACM (2003)
5. Briest, P.: Uniform budgets and the envy-free pricing problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 808–819. Springer, Heidelberg (2008)
6. Briest, P., Krysta, P.: Single-minded unlimited supply pricing on sparse instances. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, pp. 1093–1102. ACM (2006)
7. Chalermsook, P., Chuzhoy, J., Kannan, S., Khanna, S.: Improved hardness results for profit maximization pricing problems with unlimited supply. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX 2012 and RANDOM 2012. LNCS, vol. 7408, pp. 73–84. Springer, Heidelberg (2012)
8. Chen, N., Deng, X.: Envy-free pricing in multi-item markets. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 418–429. Springer, Heidelberg (2010)
9. Cole, R., Dobzinski, S., Fleischer, L.K.: Prompt mechanisms for online auctions. In: Monien, B., Schroeder, U.-P. (eds.) SAGT 2008. LNCS, vol. 4997, pp. 170–181. Springer, Heidelberg (2008)

10. Dobzinski, S., Nisan, N., Schapira, M.: Truthful randomized mechanisms for combinatorial auctions. Journal of Computer and System Sciences 78(1), 15–25 (2012)
11. Fiat, A., Wingarten, A.: Envy, multi envy, and revenue maximization. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 498–504. Springer, Heidelberg (2009)
12. Guruswami, V., Hartline, J.D., Karlin, A.R., Kempe, D., Kenyon, C., McSherry, F.: On profit-maximizing envy-free pricing. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1164–1173. Society for Industrial and Applied Mathematics (2005)
13. Im, S., Lu, P., Wang, Y.: Envy-free pricing with general supply constraints. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 483–491. Springer, Heidelberg (2010)
14. Lavi, R., Nisan, N.: Competitive analysis of incentive compatible on-line auctions. In: Proceedings of the 2nd ACM Conference on Electronic Commerce, pp. 233–241. ACM (2000)
15. Lehmann, B., Lehmann, D., Nisan, N.: Combinatorial auctions with decreasing marginal utilities. Games and Economic Behavior 55(2), 270–296 (2006)
16. Yao, A.C.C.: Probabilistic computations: Toward a unified measure of complexity. In: FOCS (1977)
17. Zhang, Y., Chin, F.Y.L., Ting, H.-F.: Competitive algorithms for online pricing. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 391–401. Springer, Heidelberg (2011)
18. Zhang, Y., Chin, F.Y.L., Ting, H.-F.: Online pricing for multi-type of items. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) AAIM 2012 and FAW 2012. LNCS, vol. 7285, pp. 82–92. Springer, Heidelberg (2012)
19. Zhang, Y., Wang, Y., Chin, F.Y.L., Ting, H.-F.: Competitive algorithms for online pricing. Discrete Mathematics, Algorithms and Applications 4(02) (2012)

RAM-Efficient External Memory Sorting

Lars Arge[1,⋆] and Mikkel Thorup[2,⋆⋆]

[1] MADALGO[⋆⋆⋆], Aarhus University, Aarhus, Denmark
[2] University of Copenhagen,[†] Copenhagen, Denmark

Abstract. In recent years a large number of problems have been considered in external memory models of computation, where the complexity measure is the number of blocks of data that are moved between slow external memory and fast internal memory (also called I/Os). In practice, however, internal memory time often dominates the total running time once I/O-efficiency has been obtained. In this paper we study algorithms for fundamental problems that are simultaneously I/O-efficient and internal memory efficient in the RAM model of computation.

1 Introduction

In the last two decades a large number of problems have been considered in the external memory model of computation, where the complexity measure is the number of blocks of elements that are moved between external and internal memory. Such movements are also called I/Os. The motivation behind the model is that random access to external memory, such as disks, often is many orders of magnitude slower than random access to internal memory; on the other hand, if external memory is accessed sequentially in large enough blocks, then the cost per element is small. In fact, disk systems are often constructed such that the time spent on a block access is comparable to the time needed to access each element in a block in internal memory.

Although the goal of external memory algorithms is to minimize the number of costly blocked accesses to external memory when processing massive datasets, it is also clear from the above that if the internal processing time per element in a block is large, then the practical running time of an I/O-efficient algorithm is dominated by internal processing time. Often I/O-efficient algorithms are in fact not only efficient in terms of I/Os, but can also be shown to be internal memory efficient in the comparison model. Still, in many cases the practical running time of I/O-efficient algorithms is dominated by the internal computation time. Thus both from a practical and a theoretical point of view it is interesting to investigate

⋆ Supported in part by the Danish National Research Foundation and the Danish National Advanced Technology Foundation.

⋆⋆ Supported in part by an Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career program.

⋆⋆⋆ Center for Massive Data Algorithmics—a center of the Danish National Research Foundation.

† Part of this work was done while the author was at AT&T Labs–Research.

how internal-memory efficient algorithms can be obtained while simultaneously ensuring that they are I/O-efficient. In this paper we consider algorithms that are both I/O-efficient and efficient in the RAM model in internal memory.

Previous results. We will be working in the standard external memory model of computation, where M is the number of elements that fit in main memory and an I/O is the process of moving a block of B consecutive elements between external and internal memory [1]. We assume that $N \geq 2M$, $M \geq 2B$ and $B \geq 2$. Computation can only be performed on elements in main memory, and we will assume that each element consists of one word. We will sometime assume the comparison model in internal memory, that is, that the only computation we can do on elements are comparisons. However, most of the time we will assume the RAM model in internal memory. In particular, we will assume that we can use elements for addressing, e.g. trivially implementing permuting in linear time. Our algorithms will respect the standard so-called *indivisibility assumption*, which states that at any given time during an algorithm the original N input elements are stored somewhere in external or internal memory. Our internal memory time measure is simply the number of performed operations; note that this includes the number of elements transferred between internal and external memory.

Aggarwal and Vitter [1] described sorting algorithms using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os. One of these algorithms, external merge-sort, is based on $\Theta(M/B)$-way merging. First $O(N/M)$ sorted runs are formed by repeatedly sorting M elements in main memory, and then these runs are merged together $\Theta(M/B)$ at a time to form longer runs. The process continues for $O(\log_{M/B} \frac{N}{M})$ phases until one is left with one sorted list. Since the initial run formation and each phase can be performed in $O(N/B)$ I/Os, the algorithm uses $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os. Another algorithm, external distribution-sort, is based on $\Theta(\sqrt{M/B})$-way splitting. The N input elements are first split into $\Theta(\sqrt{M/B})$ sets of roughly equal size, such that the elements in the first set are all smaller than the elements in the second set, and so on. Each of the sets are then split recursively. After $O(\log_{\sqrt{M/B}} \frac{N}{M}) = O(\log_{M/B} \frac{N}{M})$ split phases each set can be sorted in internal memory. Although performing the split is somewhat complicated, each phase can still be performed in $O(N/B)$ I/Os. Thus also this algorithm uses $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os.

Aggarwal and Vitter [1] proved that external merge- and distribution-sort are I/O-optimal when the comparison model is used in internal memory, and in the following we will use $sort_E(N)$ to denote the number of I/Os *per block of elements* of these optimal algorithms, that is, $sort_E(N) = O(\log_{M/B} \frac{N}{B})$ and external comparison model sort takes $\Theta(\frac{N}{B} sort_E(N))$ I/Os. (As described below, the I/O-efficient algorithms we design will move $O(N \cdot sort_E(N))$ elements between internal and external memory, so $O(sort_E(N))$ will also be the per element internal memory cost of obtaining external efficiency.) When no assumptions other than the indivisibility assumption are made about internal memory computation (i.e. covering our definition of the use of the RAM model in internal memory), Aggarwal and Vitter [1] proved that permuting N elements according to a given permutation requires $\Omega(\min\{N, \frac{N}{B} sort_E(N)\})$ I/Os. Thus this is also a lower bound for RAM model sorting. For all practical values of N, M and B the bound is $\Omega(\frac{N}{B}sort_E(N))$. Subsequently, a large number of I/O-efficient algorithms have been developed. Of particular relevance for this paper, several priority queues have been developed where insert and deletemin operations can be performed in $O(\frac{1}{B}sort_E(N))$ I/Os amortized [2,4,8]. The structure by Arge [2] is based on the so-called buffer-tree technique, which uses $O(M/B)$-way splitting, whereas the other structures also use $O(M/B)$-way merging.

In the RAM model the best known sorting algorithm uses $O(N \log\log N)$ time [6]. Similar to the I/O-case, we use $sort_I(N) = O(\log\log N)$ to denote the *per element* cost of the best known sorting algorithm. If randomization is allowed then this can be improved to $O(\sqrt{\log\log n})$ expected time [7]. A priority queue can also be implemented so that the cost per operation is $O(sort_I(N))$ [9].

Our results. In Section 2 we first discuss how both external merge-sort and external distribution-sort can be implemented to use optimal $O(N \log N)$ time if the comparison model is used in internal memory, by using an $O(N \log N)$ sorting algorithm and (in the merge-sort case) an $O(\log N)$ priority queue. We also show how these algorithms can relatively easily be modified to use

$$O(N \cdot (sort_I(N) + sort_I(M/B) \cdot sort_E(N))) \text{ and}$$

$$O(N \cdot (sort_I(N) + sort_I(M) \cdot sort_E(N)))$$

time, respectively, if the RAM model is used in internal memory, by using an $O(N \cdot sort_I(N))$ sorting algorithm and an $O(sort_I(N))$ priority queue.

The question is of course if the above RAM model sorting algorithms can be improved. In Section 2 we discuss how it seems hard to improve the running time of the merge-sort algorithm, since it uses a priority queue in the merging step. By using a linear-time internal-memory splitting algorithm, however, rather than an $O(N \cdot sort_I(N))$ sorting algorithm, we manage to improve the running time of external distribution-sort to

$$O(N \cdot (sort_I(N) + sort_E(N))).$$

Our new *split-sort* algorithm still uses $O(\frac{N}{B}sort_E(N))$ I/Os. Note that for small values of M/B the $N \cdot sort_E(N)$-term, that is, the time spent on moving elements between internal and external memory, dominates the internal time. Given the conventional wisdom that merging is superior to splitting in external memory, it is also surprising that a distribution algorithm outperforms a merging algorithm.

In Section 3 we develop an I/O-efficient RAM model priority queue by modifying the buffer-tree based structure of Arge [2]. The main modification consists of removing the need for sorting of $O(M)$ elements every time a so-called buffer-emptying process is performed. The structure supports insert and deletemin operations in $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_I(N) + sort_E(N))$ time. Thus it can be used to develop another $O(\frac{N}{B}sort_E(N))$ I/O and $O(N \cdot (sort_I(N) + sort_E(N)))$ time sorting algorithm.

Finally, in Section 4 we show that when $\frac{N}{B}sort_E(N) = o(N)$ (and our sorting algorithms are I/O-optimal), any I/O-optimal sorting algorithm must transfer a number of elements between internal and external memory equal to $\Theta(B)$ times the number of I/Os it performs, that is, it must transfer $\Omega(N \cdot sort_E(N))$ elements and thus also use $\Omega(N \cdot sort_E(N))$ internal time. In fact, we show a lower bound on the number of I/Os needed by an algorithm that transfers $b \leq B$ elements on the average per I/O, significantly extending the lower bound of Aggarwal and Vitter [1]. The result implies that (in the practically realistic case) when our split-sort and priority queue sorting algorithms are I/O-optimal, they are in fact also CPU optimal in the sense that their running time is the sum of an unavoidable term and the time used by the best known RAM sorting algorithm. As mentioned above, the lower bound also means that the time spent on moving elements between internal and external memory resulting from the fact that we are considering I/O-efficient algorithms can dominate the internal computation time, that is, considering I/O-efficient algorithms implies that less internal-memory efficient algorithms can be obtained than if not considering I/O-efficiency. Furthermore, we show that when $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$ (the *tall cache assumption*) the same $\Omega(N \cdot sort_E(N))$ number of transfers are needed for any algorithm using less than $\varepsilon N/4$ I/Os (even if it is not I/O-optimal).

To summarize our contributions, we open up a new area of algorithms that are both RAM-efficient and I/O-efficient. The area is interesting from both a theoretical and practical point of view. We illustrate that existing algorithms, in particular multiway merging based algorithms, are not RAM-efficient, and develop a new sorting algorithm that is both efficient in terms of I/O and RAM time, as well as a priority queue that can be used in such an efficient algorithm. We prove a lower bound that shows that our algorithms are both I/O and internal-memory RAM model optimal. The lower bound significantly extends the Aggarwal and Vitter lower bound [1], and shows that considering I/O-efficient algorithms influences how efficient internal-memory algorithms can be obtained.

2 Sorting

External merge-sort. In external merge-sort $\Theta(N/M)$ sorted runs are first formed by repeatedly loading M elements into main memory, sorting them, and writing them back to external memory. In the first merge phase these runs are merged together $\Theta(M/B)$ at a time to form longer runs. The merging is continued for $O(\log_{M/B} \frac{N}{M}) = O(sort_E(N))$ merge phases until one is left with one sorted run. It is easy to realize that M/B runs can be merged together in $O(N/B)$ I/Os: We simply load the first block of each of the runs into main memory, find and output the B smallest elements, and continue this process while loading a new block from the relevant run every time all elements in main memory from that particular run have been output. Thus external merge-sort uses $O(\frac{N}{B} \log_{M/B} \frac{N}{M}) = O(\frac{N}{B} sort_E(N))$ I/Os.

In terms of internal computation time, the initial run formation can trivially be performed in $O(N/M \cdot M \log M) = O(N \log M)$ time using any $O(N \log N)$ internal sorting algorithm. Using an $O(\log(M/B))$ priority queue to hold the minimal element from each of the M/B runs during a merge, each of the $O(\log_{M/B} \frac{N}{M})$ merge phases can be performed in $O(N \log \frac{M}{B})$ time. Thus external merge-sort can be implemented to use $O(N \log M + \log_{M/B} \frac{N}{M} \cdot N \log \frac{M}{B}) = O(N \log M + N \log \frac{N}{M}) = O(N \log N)$ time, which is optimal in the comparison model.

When the RAM model is used in internal memory, we can improve the internal time by using a RAM-efficient $O(M \cdot sort_I(M))$ algorithm in the run formation phase and by replacing the $O(\log(M/B))$ priority queue with an $O(sort_I(M/B))$ time priority queue [9]. This leads to an $O(N \cdot (sort_I(M) + sort_I(M/B) \cdot sort_E(N))$ algorithm. There seems no way of avoiding the extra $sort_I(M/B)$-term, since that would require an $O(1)$ priority queue.

External distribution-sort. In external distribution-sort the input set of N elements is first split into $\sqrt{M/B}$ sets $X_0, X_1, \ldots, X_{\sqrt{M/B}-1}$ defined by $s = \sqrt{M/B} - 1$ split elements $x_1 < x_2 < \ldots < x_s$, such that all elements in X_0 are smaller than x_1, all elements in $X_{\sqrt{M/B}-1}$ are larger than or equal to x_s, and such that for $1 \leq i \leq \sqrt{M/B} - 2$ all elements in X_i are larger than or equal to x_i and smaller than x_{i+1}. Each of these sets is recursively split until each set is smaller than M (and larger than $M/(M/B) = B$) and can be sorted in internal memory. If the s split elements are chosen such that $|X_i| = O(N/s)$ then there are $O(\log_s \frac{N}{B}) = O(\log_{M/B} \frac{N}{B}) = O(sort_E(N))$ split phases. Aggarwal and Vitter [1] showed how to compute a set of s split elements with this property in $O(N/B)$ I/Os. Since the actual split of the elements according to the split elements can also be performed in $O(N/B)$ I/Os (just like merging of M/B sorted runs), the total number of I/Os needed by distribution-sort is $O(\frac{N}{B} sort_E(N))$.

Ignoring the split element computation it is easy to implement external distribution-sort to use $O(N \log N)$ internal time in the comparison model: During a split we simply hold the split elements in main memory and perform a binary search among them with each input element to determine to which set X_i the element should go. Thus each of the $O(\log_{M/B} \frac{N}{B})$ split phases uses $O(N \log \sqrt{M/B})$ time. Similarly, at the end of the recursion we sort $O(N/M)$ memory loads using $O(N \log M)$ time in total. The split element computation algorithm of Aggarwal and Vitter [1], or rather its analysis, is somewhat complicated. Still it is easy to realize that it also works in $O(N \log M)$ time as required to obtain an $O(N \log N)$ time algorithm in total. The algorithm works by loading the N elements a memory load at a time, sorting them and picking every $\sqrt{M/B}/4$'th element in the sorted order. This obviously requires $O(N/M \cdot M \log M) = O(N \log M)$ time and results in a set of $4N/\sqrt{M/B}$ elements. Finally, a linear I/O and time algorithm is used $\sqrt{M/B}$ times on this set of elements to obtain the split elements, thus using $O(N)$ additional time.

If we use a RAM sorting algorithm to sort the memory loads at the end of the split recursion, the running time of this part of the algorithm is reduced to $O(N \cdot sort_I(M))$. Similarly, we can use the RAM sorting algorithm in the split element computation algorithm, resulting in an $O(N \cdot sort_I(M))$ algorithm and consequently a $sort_I(M)$-term in the total running time. Finally, in order to avoid the binary search over $\sqrt{M/B}$ split elements in the actual split algorithm, we can modify it to use sorting instead: To split N elements among s splitting elements stored in s/B blocks in main memory, we allocate a buffer of one block in main memory for each of the $s+1$ output sets. Thus in total we require $s/B + (s+1)B < M/2$ of the main memory for split elements and buffers. Next we repeatedly bring $M/2$ elements onto main memory, sort them, and distribute them to the $s+1$ buffers, while outputting the B elements in a buffer when it runs full. Thus this process requires $O(N \cdot sort_I(M))$ time and $O(N/B)$ I/Os like the split element finding algorithm. Overall this leads to an $O(N \cdot (sort_I(M) + sort_I(M) \cdot sort_E(N)))$ time algorithm.

Split-sort. While it seems hard to improve the RAM running time of the external merge-sort algorithm, we can actually modify the external distribution-sort algorithm further and obtain an algorithm that in most cases is optimal both in terms of I/O and time. This *split-sort* algorithm basically works like the distribution-sort algorithm with the split algorithm modification described above. However, we need to modify the algorithm further in order to avoid the $sort_I(M)$-term in the time bound that appears due to the repeated sorting of $O(M)$ elements in the split element finding algorithm, as well as in the actual split algorithm.

First of all, instead of sorting each batch of $M/2$ elements in the split algorithm to split them over $s = \sqrt{M/B} - 1 < \sqrt{M/2}$ split elements, we use a previous result that shows that we can actually perform the split in linear time.

Lemma 1 (Han and Thorup [7]). *In the RAM model N elements can be split over $N^{1-\varepsilon}$ split elements in linear time and space for any constant $\varepsilon > 0$.*

Secondly, in order to avoid the sorting in the split element finding algorithm of Aggarwal and Vitter [1], we design a new algorithm that finds the split elements on-line as part of the actual split algorithm, that is, we start the splitting with no split elements at all and gradually add at most $s = \sqrt{M/B} - 1$ split elements one at a time. An online split strategy was previously used by Frigo et al [5] in a cache-oblivious algorithm setting. More precisely, our algorithm works as follows. To split N input elements we, as previously, repeatedly bring $M/2$ elements onto main memory, distribute them to buffers using the current split elements and Lemma 1, while outputting the B elements in a buffer when it runs full. However, during the process we keep track of how many elements are output to each subset. If the number of elements in a subset X_i becomes $2N/s$ we pause the split algorithm, compute the median of X_i and add it to the set of splitters, and split X_i at the median element into two sets of size N/s. Then we continue the splitting algorithm.

It is easy to see that the above splitting process results in at most $s+1$ subsets containing between N/s and $2N/s - 1$ elements each, since a set is split when it has $2N/s$ elements and each new set (defined by a new split element) contains at least N/s elements. The actual median computation and the split of X_i can be performed in $O(|X_i|) = O(N/s)$ time and $O(|X_i|/B) = O(N/sB)$ I/Os [1]. Thus if we charge this cost to the at least N/s elements that were inserted in X_i

since it was created, each element is charged $O(1)$ time and $O(1/B)$ I/Os. Thus each distribution phase is performed in linear time and $O(N/B)$ I/Os, leading to an $O(N \cdot (sort_I(M) + sort_E(N)))$ time algorithm.

Theorem 1. *The split-sort algorithm can be used to sort N elements in $O(N \cdot (sort_I(M) + sort_E(N)))$ time and $O(\frac{N}{B} sort_E(N))$ I/Os.*

Remarks. Since $sort_I(M) + sort_E(N) \geq sort_I(N)$ our split-sort algorithm uses $\Omega(N \cdot sort_I(N))$ time. In Section 4 we prove that the algorithm in some sense is optimal both in terms of I/O and time. Furthermore, we believe that the algorithm is simple enough to be of practical interest.

3 Priority Queue

In this section we discuss how to implement an I/O- and RAM-efficient priority queue by modifying the I/O-efficient buffer tree priority queue [2].

Structure. Our external priority queues consists of a fanout $\sqrt{M/B}$ B-tree [3] T over $O(N/M)$ leaves containing between $M/2$ and M elements each. In such a tree, all leaves are on the same level and each node (except the root) has fan-out between $\frac{1}{2}\sqrt{M/B}$ and $\sqrt{M/B}$ and contains at most $\sqrt{M/B}$ splitting elements defining the element ranges of its children. Thus T has height $O(\log_{\sqrt{M/B}} \frac{N}{M}) = O(sort_E(N))$. To support insertions efficiently in a "lazy" manner, each internal node is augmented with a *buffer* of size M and an *insertion buffer* of size at most B is maintained in internal memory. To support deletemin operations efficiently, a RAM-efficient priority queue [9] supporting both deletemin and deletemax,[1] called the *mini-queue*, is maintained in main memory containing the up to $M/2$ smallest elements in the priority queue.

Insertion. To perform an insertion we first check if the element to be inserted is smaller than the maximal element in the mini-queue, in which case we insert the new element in the mini-queue and continue the insertion process with the currently maximal element in the mini-queue. Next we insert the element to be inserted in the insertion buffer. When we have collected B elements in the insertion buffer we insert them in the buffer of the root. If this buffer now contains more than $M/2$ elements we perform a *buffer-emptying process* on it, "pushing" elements in the buffer one level down to buffers on the next level of T: We load the $M/2$ oldest elements into main memory along with the less than $\sqrt{M/B}$ splitting elements, distribute the elements among the splitting elements, and finally output them to the buffers of the relevant children. Since the splitting and buffer elements fit in memory and the buffer elements are distributed to $\sqrt{M/B}$ buffers one level down, the buffer-emptying process is performed in $O(M/B)$ I/Os. Since we distribute $M/2$ elements using $\sqrt{M/B}$ splitters the process can

[1] A priority queue supporting both deletemin and deletemax can easily be obtained using two priority queues supporting deletemin and delete as the one by Thorup [9].

be performed in $O(M)$ time (Lemma 1). After emptying the buffer of the root some of the nodes on the next level may contain more than $M/2$ elements. If they do we perform recursive buffer-emptying processes on these nodes. Note that this way buffers will never contain more than M elements. When (between 1 and $M/2$) elements are pushed down to a leaf (when performing a buffer-emptying process on its parent) resulting in the leaf containing more than M (and less than $3M/2$) elements we split it into two leaves containing between $M/2$ and $3M/4$ elements each. We can easily do so in $O(M/B)$ I/Os and $O(M)$ time [1]. As a result of the split the parent node v gains a child, that is, a new leaf is inserted. If needed, T is then balanced using node splits as a normal B-tree, that is, if the parent node now has $\sqrt{M/B}$ children it is split into two nodes with $1/2\sqrt{M/B}$ children each, while also distributing the elements in v's buffer among the two new nodes. This can easily be accomplished in $O(M/B)$ I/Os and M time. The rebalancing may propagate up along the path to the root (when the root splits a new root with two children is constructed).

During buffer-emptying processes we push $\Theta(M)$ elements one level down the tree using $O(M/B)$ I/Os and $O(M)$ time. Thus each element inserted in the root buffer pays $O(1/B)$ I/Os and $O(1)$ time amortized, or $O(\frac{1}{B}\log_{M/B}\frac{N}{B}) = O(\frac{1}{B}sort_E(N))$ I/Os and $O(\log_{M/B}\frac{N}{B}) = O(sort_E(N))$ time amortized on buffer-emptying processes on a root-leaf path. When a leaf splits we may use $O(M/B)$ I/Os and $O(M)$ time in each node of a leaf-root path of length $O(sort_E(N))$. Amortizing among the at least $M/4$ elements that were inserted in the leaf since it was created, each element is charged and additional $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N))$ time on insertion in the root buffer. Since insertion of an element in the root buffer is always triggered by an insertion operation, we can charge the $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N))$ time cost to the insertion operation.

Deletemin. To perform a deletemin operation we first check if the mini-queue contains any elements. If it does we simply perform a deletemin operation on it and return the retrieved element using $O(sort_I(M))$ time and no I/Os. Otherwise we perform buffer-emptying processes on all nodes on the leftmost path in T starting at the root and moving towards the leftmost leaf. After this the buffers on the leftmost path are all empty and the smallest elements in the structure are stored in the leftmost leaf. We load the between $M/2$ and M elements in the leaf into main memory, sort them and remove the smallest $M/2$ elements and insert them in the mini-queue in internal memory. If this results in the leaf having less than $M/2$ elements we insert the elements in a sibling and delete the leaf. If the sibling now has more than M elements we split it. As a result of this the parent node v may lose a child. If needed T is then rebalanced using node fusions as a normal B-tree, that is, if v now has $1/2\sqrt{M/B}$ children it is fused with its sibling (possibly followed by a split). As with splits after insertion of a new leaf, the rebalancing may propagate up along the path to the root (when the root only has one leaf left it is removed). Note that no buffer merging is needed since the buffers on the leftmost path are all empty.

If buffer-emptying processes are needed during a deletemin operation we spend $O(\frac{M}{B}\log_{M/B}\frac{N}{B}) = O(\frac{M}{B} sort_E(N))$ I/Os and $O(M\log_{M/B}\frac{N}{B}) = O(M \cdot sort_E(N))$ time on such processes that are not paid by buffers running full (containing more than $M/2$ elements). We also use $O(M/B)$ I/Os and $O(M \cdot sort_I(M))$ time to load and sort the leftmost leaf, and another $O(M \cdot sort_I(M))$ time is used to insert the $M/2$ smallest elements in the mini-queue. Then we may spend (M/B) I/Os and $O(M)$ time on each of at most $O(\log_{M/B}\frac{N}{B})$ nodes on the leftmost path that need to be fused or split. Altogether the filling up of the mini-queue requires $O(\frac{M}{B} sort_E(N))$ I/Os and $O(M \cdot (sort_I(M) + sort_E(N)))$ time. Since we only fill up the mini-queue when $M/2$ deletemin operations have been performed since the last fill up, we can amortize this cost over these $M/2$ deletemin operations such that each deletemin is charged $O(\frac{1}{B} sort_E(N))$ I/Os and $O(sort_E(N) + sort_I(M))$ time.

Theorem 2. *There exists a priority queue supporting an insert operation in $O(\frac{1}{B} sort_E(N))$ I/Os and $O(sort_E(N))$ time amortized and a deletemin operation in $O(\frac{1}{B} sort_E(N))$ I/Os and $O(sort_I(M) + sort_E(N))$ time amortized.*

Remarks. Our priority queue obviously can be used in a simple $O(\frac{N}{B} sort_E(N))$ I/O and $O(N \cdot (sort_I(M) + sort_E(N)))$ time sorting algorithm. Note that it is essential that a buffer-emptying process does not require sorting of the elements in the buffer. In normal buffer-trees [2] such a sorting is indeed performed, mainly to be able to support deletions and (batched) rangesearch operations efficiently. Using a more elaborate buffer-emptying process we can also support deletions without the need for sorting of buffer elements.

4 Lower Bound

Assume that $\frac{N}{B} sort_E(N) = o(N)$ and for simplicity also that B divides N. Recall that under the indivisibility assumption we assume the RAM model in internal memory but require that at any time during an algorithm the original N elements are stored somewhere in memory; we allow copying of the original elements. The internal memory contains at most M elements and the external memory is divided into N *blocks* of B elements each; we only need to consider N blocks, since we are considering algorithms doing less than N I/Os. During an algorithm, we let X denote the set of original elements (including copies) in internal memory and Y_i the set of original elements (including copies) in the i'th block; an I/O transfers *up to* B elements between an Y_i and X. Note that in terms of CPU time, an I/O can cost anywhere between 1 and B (transfers).

In the external memory permuting problem, we are given N elements in the first N/B blocks and want to rearrange them according to a given permutation; since we can always rearrange the elements within the N/B blocks in $O(N/B)$ I/Os, a permutation is simply given as an assignment of elements to blocks (i.e. we ignore the order of the elements within a block). In other words, we start with a distribution of N elements in $X, Y_1, Y_2, \ldots Y_N$ such that $|Y_1| = |Y_2| = \ldots = |Y_{N/B}| = B$ and $X = Y_{(N/B)+1} = Y_{(N/B)+2} = \ldots = Y_N = \emptyset$, and should produce another given distribution of the same elements such that $|Y_1| = |Y_2| = \ldots = |Y_{N/B}| = B$ and $X = Y_{(N/B)+1} = Y_{(N/B)+2} = \ldots = Y_N = \emptyset$.

To show that any permutation algorithm that performs $O(\frac{N}{B} sort_E(N))$ I/Os has to transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory, we first note that at any given time during a permutation algorithm we can identify a distribution (or more) of the original N elements (or copies of them) in $X, Y_1, Y_2, \ldots Y_N$. We then first want to bound the number of distributions that can be created using T I/Os, given that b_i, $1 \leq i \leq T$, is the number of elements transferred in the i'th I/O; any correct permutation algorithm needs to be able to create at least $\frac{N!}{B!^{N/B}} = \Omega((N/B)^N)$ distributions.

Consider the i'th I/O. There are at most N possible choices for the block Y_j involved in the I/O; the I/O either transfers $b_i \leq B$ elements from X to Y_j or from Y_j to X. In the first case there are at most $\binom{M}{b_i}$ ways of choosing the b_i elements, and each element is either moved or copied. In the second case there are at most most $\binom{B}{b_i}$ ways of choosing the elements to move or copy. Thus the I/O can at most increase the number of distributions that can be created by a factor of

$$N \cdot \left(\binom{M}{b_i} + \binom{B}{b_i} \right) \cdot 2^{b_i} < N(2eM/b_i)^{2b_i}.$$

Now the T I/Os can thus at most create $\prod_{i=1}^{T} N(2eM/b_i)^{2b_i}$ distributions. That this number is bounded by $\left(N(2eM/b)^{2b}\right)^T$, where b is the average of the b_i's, can be seen by just considering two values b_1 and b_2 with average b. In this case we have

$$N(2eM/b_1)^{2b_1} \cdot N(2eM/b_2)^{2b_2} \leq \frac{N^2(2eM)^{2(b_1+b_2)}}{b^{2(b_1+b_2)}} \leq \left(N(2eM/b)^{2b}\right)^2.$$

Next we consider the number of distributions that can be created using T I/Os for all possible values of b_i, $1 \leq i \leq T$, with a given average b. This can trivially be bounded by multiplying the above bound by B^T (since this is a bound on the total number of possible sequences $b_1, b_2, \ldots, b_T$). Thus the number of distributions is bounded by $B^T \left(N(2eM/b)^{2b}\right)^T = ((BN)(2eM/b)^{2b})^T$. Since any permutation algorithm needs to be able to create $\Omega((N/B)^N)$ distributions, we get the following lower bound on the number of I/Os $T(b)$ needed by an algorithm that transfers $b \leq B$ elements on the average per I/O:

$$T(b) = \Omega\left(\frac{N \log(N/B)}{\log N + b \log(M/b)}\right).$$

Now $T(B) = \Omega(\min\{N, \frac{N}{B} sort_E(N)\})$ corresponds to the lower bound proved by Aggarwal and Vitter [1]. Thus when $\frac{N}{B} sort_E(N) = o(N)$ we get $T(B) = \Omega(\frac{N}{B} sort_E(N)) = \Omega\left(\frac{N \log(N/B)}{B \log(M/B)}\right)$. Since $1 \leq b \leq B \leq M/2$, we have $T(b) = \omega(T(B))$ for $b = o(B)$. Thus any algorithm performing optimal $O(\frac{N}{B} sort_E(N))$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory.

Reconsider the above analysis under the tall cache assumption $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$. In this case, we have that the number of distributions any permutation algorithm needs to be able to create is $\Omega((N/B)^N) = \Omega(N^{\varepsilon N})$. Above we proved that with T I/Os transferring an average number of b keys an algorithm can create at most $(BN(2eM/b)^{2b})^T < N^{2T}M^{2bT}$ distributions. Thus we have $M^{2bT} \geq N^{\varepsilon N - 2T}$. For $T < \varepsilon N/4$, we get $M^{2bT} \geq N^{\varepsilon N/2}$ and thus that the number of transferred elements bT is $\Omega(N \log_M N)$. Since the tall cache assumption implies that $\log(N/B) = \Theta(\log N)$ and $\log(M/B) = \Theta(\log M)$ we have that $N \log_M N = \Theta(N \log_{M/B}(N/B)) = \Theta(N \cdot sort_E(N))$. Thus any algorithm using less than $\varepsilon N/4$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory.

Theorem 3. *When $B \leq \frac{1}{2}M$ and $\frac{N}{B} sort_E(N) = o(N)$, any I/O-optimal permuting algorithm must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory under the indivisibility assumption.*

When $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$ any, permuting algorithm using less than $\varepsilon N/4$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory under the indivisibility assumption.

Remark. The above means that in practice where $\frac{N}{B} sort_E(N) = o(N)$ our $O(\frac{N}{B} sort_E(N))$ I/O and $O(N \cdot (sort_I(N) + sort_E(N))$ time split-sort and priority queue sort algorithms are not only I/O-optimal but also CPU optimal in the sense that their running time is the sum of an unavoidable term and the time used by the best known RAM sorting algorithm.

References

1. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Communications of the ACM 31(9), 1116–1127 (1988)
2. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37(1), 1–24 (2003)
3. Comer, D.: The ubiquitous B-tree. ACM Computing Surveys 11(2), 121–137 (1979)
4. Fadel, R., Jakobsen, K.V., Katajainen, J., Teuhola, J.: Heaps and heapsort on secondary storage. Theoretical Computer Science 220(2), 345–362 (1999)
5. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. IEEE Symposium on Foundations of Computer Science, pp. 285–298 (1999)
6. Han, Y.: Deterministic sorting in $O(n \log \log n)$ time and linear space. In: Proc. ACM Symposium on Theory of Computation, pp. 602–608 (2002)
7. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In: Proc. IEEE Symposium on Foundations of Computer Science, pp. 135–144 (2002)
8. Kumar, V., Schwabe, E.: Improved algorithms and data structures for solving graph problems in external memory. In: Proc. IEEE Symp. on Parallel and Distributed Processing, pp. 169–177 (1996)
9. Thorup, M.: Equivalence between priority queues and sorting. Journal of the ACM 54(6), Article 28 (2007)

Succinct Data Structures for Representing Equivalence Classes*

Moshe Lewenstein[1], J. Ian Munro[2], and Venkatesh Raman[3]

[1] Department of Computer Science, Bar Ilan University, Israel
moshe@cs.biu.ac.il

[2] Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

[3] The Institute of Mathematical Sciences, Chennai, India
vraman@imsc.res.in

Abstract. Given a partition of an n element set into equivalence classes, we consider time-space tradeoffs for representing it to support the query that asks whether two given elements are in the same equivalence class. This has various applications including for testing whether two vertices are in the same connected component in an undirected graph or in the same strongly connected component in a directed graph.

We consider the problem in several models.

– Concerning labeling schemes where we assign labels to elements and the query is to be answered just by examining the labels of the queried elements (without any extra space): if each vertex is required to have a unique label, then we show that a label space of $\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$ is necessary and sufficient. In other words, $\lg n + \lg\lg n + O(1)$ bits of space are necessary and sufficient for representing each of the labels. This slightly strengthens the known lower bound and is in contrast to the known necessary and sufficient bound of $\lceil \lg n \rceil$ for the label length, if each vertex need not get a unique label.

– Concerning succinct data structures for the problem when the n elements are to be uniquely assigned labels from label set $\{1, \ldots, n\}$, we first show that $\Theta(\sqrt{n})$ bits are necessary and sufficient to represent the equivalence class information. This space includes the space for implicitly encoding the vertex labels. We can support the query in such a structure in $O(\lg n)$ time in the standard word RAM model. We then develop structures where the queries can be answered
  - in $O(\lg\lg n)$ time using $O(\sqrt{n}\lg n/\lg\lg n)$ bits, and
  - in $O(1)$ time using $O(\sqrt{n}\lg n)$ bits of space.

We also develop a dynamic structure that uses $O(\sqrt{n}\lg n)$ bits to support equivalence queries and unions in $O(\lg n/\lg\lg n)$ worst case time or $O(\alpha(n))$ expected amortized time where $\alpha(n)$ is the inverse Ackermann function.

* Work done while the first and the third authors were on sabbatical at the University of Waterloo, Canada; research supported by NSERC and Canada Research Chairs Programme.

1 Introduction and Motivation

We look at the following problem. Given a partition of an n element set into equivalence classes, preprocess it, assigning a unique label to each element, to obtain a data structure with minimum space to support the following query: given two elements, determine whether they are in the same equivalence class. We call the query an 'equivalence query'. This is a fundamental data structure problem that has various applications including for testing whether two vertices are in the same connected (or strongly connected) component in an undirected (or directed) graph. We study the problem in the context of succinct data structures. Designing succinct (or space efficient) data structures has been an area of interest in theory and practice motivated by the need to store large amount of data. See [4,11,19,21,8] for succinct representations of dictionaries, trees, arbitrary graphs and partially ordered sets.

We address the time-space tradeoff for representing an equivalence class and answering the equivalence query in a couple of models. Katz, Katz, Korman and Peleg [18] introduced the notion of labeling schemes whereby every node of the graph is assigned a (not necessarily distinct) label and the required query is to be answered by just looking at the labels of the query elements. They showed that $\Omega(k \lg n)$ [1] is a lower bound of the length of the label to answer 'k-connectivity queries', for k up to polylogarithmic in n. For $k = 1$ (which is the case for the problem in this paper), this lower bound is $\lceil \lg n \rceil$ and hence the scheme that simply assigns all elements of an equivalence class a single label that is distinct from the labels of other equivalence classes, is optimal in this model. However, in some situations (for example when one wants to support other graph operations including adjacency relations) we may want to give unique labels to each vertex. Our first result is that in this case, we need a label space of $\sum_{i=1}^{n} \lfloor n/i \rfloor$, and we show that this number of labels is also sufficient. We also give an encoding scheme that uses the optimal $\lg n + \lg \lg n + O(1)$ bits for the labels. The encoding scheme is similar to the one in [2], but our lower bound is stronger, and more importantly we establish an exact tight bound for the label space. This result is discussed in Section 2.

Then, in Section 3, we give succinct data structures for the problem in the model where the labels can be freely reassigned (but need to be unique and in the range 1 to n), and the query can be answered by looking at a small space data structure. We first observe that the information theoretic lower bound to represent the equivalence class information is $\Omega(\sqrt{n})$ bits, and we provide a scheme using $O(\sqrt{n})$ bits in which the query can be answered in $O(\lg n)$ time. In the rest of the section, we develop a data structure where the query can be answered in constant time albeit using $O(\sqrt{n} \lg n)$ bits of space. In Section 4, we develop methods that also support merge operation on the equivalence classes using asymptotically the same space, as fast as other known non-space efficient structures.

[1] We use $\lg n$ to denote $\log_2 n$

These structures operate in the standard word RAM model with a word size of $w = \Omega(\lg n)$ [16] where multiplication and shifts can be performed in constant time. Furthermore, our succinct structures modify the initial labels of the elements to an implicit labeling scheme. We discuss applications and limitations of this approach in Section 5.

2 Labeling Scheme with Unique Labels for Elements

In the problem, which we call the *direct equivalence queries* problem, each element is to be given a unique label, and the equivalence query is to be answered by computing directly from the two labels. It is known [2] that $\lg n + \Theta(\lg \lg n)$ bits of space are necessary and sufficient to represent the labels. We strengthen the bound to $\lg n + \lg \lg n + \Theta(1)$. The encoding that achieves this bound is similar to the one in [2], we provide it for completeness, but our lower bound establishes a tight bound on the label space. We first prove the following theorem.

Theorem 1. *Let a partition of an n element set into equivalence classes be given as input to the direct equivalence queries problem. Then a label space of $\sum_{i=1}^{n} \lfloor n/i \rfloor$ is necessary and sufficient.*

Proof. Our key observation for the sufficiency is that the i-th largest equivalence class contains at most $\lfloor n/i \rfloor$ elements. For the upper bound, we simply assign labels from the set of integers in the range $[\sum_{j=1}^{i-1}(\lfloor n/j \rfloor) + 1, \sum_{j=1}^{i} \lfloor n/j \rfloor]$ for the i-th largest equivalence class, for $i > 1$, and integers in the range $[1, n]$ for the largest equivalence class.

To show that this many labels are necessary, consider a labeling scheme for a direct equivalence queries problem. It reserves a set of labels for each equivalence class to ensure that the query is answered correctly by looking only at the labels (without knowing the equivalence classes themselves). Consider the labels assigned by such a scheme for any of the following collection of n equivalence relations (partitions of an n element set). The collection C_i contains i sets (equivalence classes) each containing $\lfloor n/i \rfloor$ or $\lceil n/i \rceil$ elements. In particular if s and t are the sizes of two of these classes, then $|s - t| \leq 1$.

Note that the labels assigned to the relation C_1 can be assigned to at most one class of each of $C_i, i = 2$ to n. This happens because every pair of elements are in the same equivalence class in C_1, and hence we will have a conflict (to answer the equivalence query looking only at the labels) if these labels are assigned to more than one class of $C_i, i = 2$ to n. Now remove C_1, and all classes from $C_i, i = 2$ to n that have been assigned the same labels as of C_1. Now the proof follows by repeating the above argument with the labels assigned to the elements of (the remaining classes of) C_2, C_3 up to C_n in that order. $\square$

To answer the equivalence query in the above labeling scheme, given an integer label x, we need to find the largest i such that $\sum_{j=1}^{i-1} \lfloor n/j \rfloor < x$. In order to support this query in constant time, we modify the labeling scheme slightly (and use space slightly suboptimal, up to lower order terms). We first order the equivalence classes in non-increasing order of their sizes. We give them labels, say 1 to c where c is the number of classes. Within each class, we give an arbitrary ordering of the elements. Then the label for an element x is given by a pair (i, j) where i is the label of the class to which the element belongs, and j is its 'rank' in the class numbered i. As the i-th largest equivalence class contains at most $\lfloor n/i \rfloor$ elements, the label j can be represented using $\lceil \lg \lfloor n/i \rfloor \rceil$ bits of space. The label i is represented using $\lceil \lg i \rceil$ bits. As the size of the representation of i is not fixed, we need to store information to find the 'break point' between i and j. Hence we 'prefix' the label (i, j) by storing the length of i in binary, using $\lceil \lg \lceil \lg n \rceil \rceil$ bits. The equivalence query can easily be answered by looking at the first component (i) of the label in constant time. The number of bits used for a label is $\lceil \lg \lceil \lg n \rceil \rceil + \lceil \lg i \rceil + \lceil \lg \lfloor n/i \rfloor \rceil$ which is at most $\lg n + \lg\lg n + 2$.

From Theorem 1, $\lceil \lg \sum_{i=1}^{n} \lfloor n/i \rfloor \rceil = \lceil \lg(n \ln n - O(n)) \rceil$ bits are necessary for the label length. Thus we have

Theorem 2. *Given a partition of an n element set into equivalence classes, we can assign to each of the elements a label of $\lg n + \lg\lg n + 2$ bits such that the equivalence query can be answered in constant time by looking only at the labels. In this model, $\lg n + \lg\lg n - \Theta(1)$ bits are necessary to represent the labels.*

3 Succinct Data Structures

Now we move on to designing data structures, where the labels of the n elements can be freely reassigned, but they need to be unique and in the range 1 to n. The queries can be answered by looking at an augmented data structure. We are interested in time and space efficient data structures. We first assign an implicit ordering of the elements. Each element gets a label according to this ordering, and the queries are answered by looking at these labels and an augmented data structure.

First, we address the question of how much space is required to capture the given equivalence class information. The information theory lower bound for the representation is given by the number of partitions of an n element set into equivalence classes, which is the same as the number of partions of n, which by the Hardy-Ramanujan formula [17] is asymptotically $\frac{1}{4n\sqrt{3}} e^{(\pi\sqrt{\frac{2n}{3}})}$. Hence the information theoretic lower bound for space to represent the equivalence class information is given by $\pi\sqrt{2n/3} \lg e - \lg n + O(1)$ which is $\Theta(\sqrt{n})$.

Now, to design space efficient data structures, let c be the number of classes, $s_i, i = 1$ to k be the distinct sizes of the classes, and let n_i be the number of classes of size s_i in the given equivalence class. Key to our structure is ordering the classes in non-decreasing order of $\gamma_i = s_i n_i$. I.e. $s_i n_i \leq s_{i+1} n_{i+1}$, for $i = 1$ to $k - 1$. We first make the simple observations that

$$\sum_{i=1}^{k} s_i n_i = n, \sum_{i=1}^{k} n_i = c \text{ and } s_i n_i \geq i \text{ for } i = 1 \text{ to } k \tag{1}$$

The last inequality follows as the i-th smallest s_i value is at least i. It follows from these observations that that $k \leq c$ and $k \leq \sqrt{2n}$.

3.1 Structure Using $O(\sqrt{n})$ Bits

Here, we design a structure that uses $O(\sqrt{n})$ bits of space to represent the equivalence class information, and can support equivalence query in $O(\lg n)$ time. Our primary structure consists of two sequences:

– the sequence s that consists of $\delta_i = s_i n_i - s_{i-1} n_{i-1}$, $i = 1$ to k, where $s_0 n_0$ is defined to be 0 and
– the sequence m that consists of $n_i, i = 1$ to k.

Each element in these sequences is represented in binary (using respectively $1 + \lceil \lg(\delta_i + 1) \rceil$ and $1 + \lceil \lg(n_i + 1) \rceil$ bits). As the lengths of each element in the sequence vary, we store two other sequences that 'shadow' the two primary sequences. The first one ψ has a 1 at the starting point of each element in the sequence s and 0 at other positions. Similarly, the second one ρ stores a 1 at the starting point of elements of the sequence m, and 0 at other positions. We also store a select structure (see for example [21,14]) on these two sequences ψ and ρ to identify the 1s quickly. The space occupied by each of these two sequences is clearly the same as that occupied by the two primary sequences, plus lower order terms.

The first sequence gives an implicit ordering of the elements, i.e. the elements in the first n_1 classes are assigned label values 1 to $s_1 n_1$, the elements of the next n_2 classes are assigned the next $s_2 n_2$ label values and so on.

We first claim that the space occupied by these four sequences is $O(\sqrt{n})$ bits. We first show the following Lemma. If any $\delta_j = 0$, then we account for 1 bit for its representation and as k is $O(\sqrt{n})$, this doesn't affect the claimed bound; so assume that $\delta_j \geq 1$ for all j in the sum below.

Lemma 1. $\sum_{j=1}^{k} \lg \delta_j$ *is* $O(\sqrt{n})$ *where each* δ_j *(as defined above) is at least* 1.

Proof. We use the following claim to achieve the desired bound.

Claim: For an integer $1 \leq i \leq n$, the number of j's such that $\delta_j \geq i$ is at most $\sqrt{2n/i}$.
Proof of claim: Let $\delta_{j_t} \geq i$, for some $t = 1$ to b. Then $s_{j_t} n_{j_t} \geq ti$, and hence

$$\sum_{t=1}^{b} ti \leq \sum_{t=1}^{b} s_{j_t} n_{j_t} \leq n$$

from which it follows that $b(b+1)/2 \leq n/i$ or $b \leq \sqrt{2n/i}$ which proves the claim. □
From the claim, it follows that (by breaking the δ values into ranges of powers of two – i.e. those between 2^{p-1} and 2^p for various values of p)

$$\sum_{j=1}^{k} \lg \delta_j \leq \sum_{p=1}^{\lceil \lg n \rceil} (\sqrt{2n/2^{(p-1)}})p = 2\sqrt{n} \sum_{p=1}^{\lceil \lg n \rceil} \frac{p}{2^{p/2}}$$

which is $O(\sqrt{n})$. □

A similar proof shows that $\sum_{j=1}^{k} \lg n_j$ is $O(\sqrt{n})$. This is because if $n_j = i$ for some j, then $s_j n_j \geq ji$, and a claim as above follows for the number of j's with $n_j = i$ as well. Thus we have a structure to represent the equivalence class information that uses $O(\sqrt{n})$ bits.

Implementing the Equivalence Query. Now, given an element labeled x, the equivalence class it belongs to is determined by first finding the predecessor $p(x)$ of x, which is $max\{j | \sum_{i=1}^{j} s_i n_i < x\}$. Given two elements x and y, if $p(x)$ and $p(y)$ are not the same, then x and y are not in the same equivalence class.

If $p(x)$ and $p(y)$ are the same, then we know that x and y are in classes that have the same sizes, but it is still not clear whether they are in the same equivalence class. They are in the same equivalence class if and only if $\lceil (x - \sum_{i=1}^{p(x)} s_i n_i)/n_{p(x)+1} \rceil$ and $\lceil (y - \sum_{i=1}^{p(y)} s_i n_i)/n_{p(y)+1} \rceil$ are the same. To compute the n_i value for some i, we simply look for the i-th and $(i+1)$-st 1 in the sequence ρ (using the select data structure on ρ) which gives the starting position and the length of the representation of n_i in the sequence m.

Now in order to support the predecessor queries in a reasonable amount of time, we store more: we simply store the $\sum_{j=1}^{i} s_j n_j$ for every value of i which is a multiple of $\lg n$. This takes $O(\sqrt{n})$ bits.

Now $p(x)$ can be obtained by doing a binary search for x on these partial sum values $\sum_{j=1}^{i} s_j n_j$ for every value of i which is a multiple of $\lg n$. Once an $O(\lg n)$ range of the predecessor is found, the actual predecessor value is found by doing a linear search on the delta values in this range. As before, the lengths and the starting positions of the δ values can be found using the select substructure on the sequence ψ. Thus we have

Theorem 3. *Given a partition of an n element set into equivalence classes, it can be stored using $O(\sqrt{n})$ bits such that the equivalence query can be answered in $O(\lg n)$ time. Furthermore, $\Omega(\sqrt{n})$ is the minimum number of bits necessary to store the equivalence class information on an n element set.*

We remark that due to Lemma 1, any gap-style encoding [7,15,9] of the δ_i and the n_i values can reduce the space by a constant factor without increasing the time.

3.2 Faster, Space-Efficient Methods

Here we develop a data structure where the equivalence query can be answered in constant time albeit using $O(\sqrt{n} \lg n)$ bits of space.

Our initial representation consists of storing

– the sequence $\sum_{j=1}^{i} s_j n_j$, $i = 1$ to k, and
– the sequence $n_i, i = 1$ to k,

where each number in each sequence is represented in binary using $\lceil \lg n \rceil$ bits. As before, the first sequence gives an implicit ordering of the elements. That is, the $s_1 n_1$ elements of the first n_1 classes form the *first* $s_1 n_1$ elements and so on. The total space used by the four sequences is at most $2\sqrt{2n}\lceil \lg n\rceil$ bits. As discussed earlier, to answer the equivalence queries, we essentially have to support predecessor queries in the sequence $\sum_{j=1}^{i} s_j n_j$, $i = 1$ to k.

A simple binary search can support the predecessor query in $O(\lg k)$ time. A y-fast trie [23] can support the predecessor query in $O(\lg\lg n)$ time. As in the scheme of the previous subsection, we could store the complete partial sums and a y-fast trie structure storing every $\lg\lg n$-th element in the partial sum sequence and store the δ values for the remaining elements of the sequence. This will help us find a range of $\lg\lg n$ for the predecessor in $O(\lg\lg n)$ time. Within the range, we can do a sequential search for the predecessor using the δ values. As the delta values require only $O(\sqrt{n})$ bits of space, we have

Theorem 4. *Given a partition of an n element set into equivalence classes, it can be stored using $O(\sqrt{n}\lg n/\lg\lg n)$ bits such that the equivalence query can be answered in $O(\lg\lg n)$ time.*

A fully indexable dictionary [21] with the improved redundancy of [14] can support the predecessor query in constant time albeit using $O(\sqrt{n}^{1+\epsilon})$ bits of space. However, we argue below that the predecessor can be supported in constant time using an additional $O(\sqrt{n}\lg n)$ bits using the fact that our sequence satisfies the last inequality in equation (1) and hence is special. In addition to the two sequences above, we store an array A of $\lceil\sqrt{2n}\rceil$ pointers, where $A[i] = max\{j | \sum_{t=1}^{j} s_t n_t \le i(i+1)/2\}$, for $i = 1$ to $\lceil\sqrt{2n}\rceil$. Now, we claim

Lemma 2. *The predecessor $p(x)$ of an integer x ($1 \le x \le n$) in the sequence $\sum_{t=1}^{i} s_t n_t, i = 1$ to k is $A[\lceil\sqrt{2x}\rceil - 1]$ or $A[\lceil\sqrt{2x}\rceil - 1] - 1$ or $A[\lceil\sqrt{2x}\rceil - 1] + 1$.*

Proof. Let $i = \lceil\sqrt{2x}\rceil - 1$, then

$$x - (\sqrt{x})/2 \le i(i+1)/2 < x + (\sqrt{x})/2,$$

and

$$x + (\sqrt{x})/2 \le (i+1)(i+2)/2 < x + 3(\sqrt{x})/2.$$

For $j = A[i]+1$, $\sum_{t=1}^{j} s_t n_t > i(i+1)/2$ (by definition of $A[i]$). Hence $s_j n_j \ge (i+1)$ and hence $s_{j+1}n_{j+1} \ge (i+2)$ hence $\sum_{t=1}^{j+1} s_t n_t > i(i+1)/2 + i + 2 > x$ and hence $p(x) \le j = A[i] + 1$.

Let $l = p(x)$. Then $\sum_{t=1}^{l+1} s_t n_t > x$ and hence $s_{l+1}n_{l+1} \ge \lceil\sqrt{2x}\rceil - 1$. Hence $\sum_{t=1}^{l+2} s_t n_t \ge x + \lceil(\sqrt{2x})\rceil > i(i+1)/2$. Hence $A[i] \le l + 1 = p(x) + 1$ which implies that $A[i] + 1 \ge p(x) \ge A[i] - 1$. □

The actual value of $p(x)$ can be computed by looking at the sum up to each of these three values.

Computing Square Roots. Note that computing $\lceil\sqrt{x}\rceil$ is not a constant time operation in the standard word RAM model. The standard Newton's iterative method uses $\Theta(\lg\lg n)$ operations. We describe a space efficient method that avoids explicit computation of square roots (for the range we are interested in) by using a look up to precomputed tables. See [20] for a similar look up table method to compute the integer closer to the square roots of integers.

Our method uses two tables, one when the number of digits of x (up to its most significant 1) is odd, denoted by O, and one when the number of digits is even, denoted by E. It turns out that $O[i]$ and $E[i]$ are quite close in value, where $E[i]$ is roughly a $\sqrt{2}$ factor larger than $O[i]$.

For $i = 1$ to $\lceil\sqrt{2n}\rceil$, we precompute and store in $E[i]$, the value of $\lceil\sqrt{i2^{(\lceil\lg(i+1)\rceil)/2}}\rceil$ and in $O[i]$, the value of $\lceil\sqrt{i2^{(\lceil\lg(i+1)\rceil)/2-1}}\rceil$. This takes $O(\sqrt{n}\lg n)$ bits. Now, given an integer i, $1 \leq i \leq 2n$, we compute $\lceil\sqrt{i}\rceil$ as follows. Let $i = a_i 2^{\lceil(\lg i)/2\rceil} + b_i$ where $b_i < 2^{\lceil(\lg i)/2\rceil}$. Then,

Lemma 3. *$\lceil\sqrt{i}\rceil = E[a_i]$ or $E[a_i+1]$ if the number of digits in i (up to its most significant 1) is even, and is $O[a_i]$ or $O[a_i+1]$ otherwise.*

Proof. As $i = a_i 2^{\lceil(\lg i)/2)\rceil} + b_i$, $a_i 2^{\lceil(\lg i)/2\rceil} \leq i < (a_i+1)2^{\lceil(\lg i)/2\rceil}$, and hence $\lceil\sqrt{a_i 2^{\lceil(\lg i)/2\rceil}}\rceil \leq \lceil\sqrt{i}\rceil \leq \lceil\sqrt{(a_i+1)2^{\lceil(\lg i)/2\rceil}}\rceil$ which is what we wanted to show. □

The actual value of $\lceil\sqrt{i}\rceil$ can be computed by squaring the values in the table and comparing them with i. Note that for $i \leq 2n$, $a_i \leq \lceil\sqrt{2n}\rceil$, and it can be obtained as follows: find the most significant bit, say bit r, mask the lower r bits to keep only the higher half of them, i.e. $\lfloor\frac{r}{2}\rfloor$ of the bits (without the leading zeroes), and finally shifting them to the right by $\lceil\frac{r}{2}\rceil$. The most significant bit can be found in constant time with the standard RAM operations, see [13]. Thus we have

Lemma 4. *For $1 \leq i \leq n$, $\lceil\sqrt{i}\rceil$ can be computed in constant time (for each i) using a precomputed table of $O(\sqrt{n}\lg n)$ bits.*

Indeed using this approach to provide a seed for Newton iteration, one can compute $\lceil\sqrt{i}\rceil$, for $i = 1$ to n in time $O(\lg(1/\epsilon))$ using a table of $O((n^\epsilon \lg n)/\epsilon)$ bits, for any positive constant $\epsilon < 1$. To summarize, we have

Theorem 5. *Given a partition of an n element set into equivalence classes, the partition can be represented using $O(\sqrt{n}\lg n)$ bits such that the equivalence query can be answered in constant time.*

4 Supporting Unions

Finally we discuss space efficient structures that can support merging of two classes in an equivalence relation and still support equivalence queries. The merge operation takes two classes of the equivalence relation and merges them to obtain a new class destroying both the old ones. We show

Theorem 6. *Given a partition of an n element set into equivalence classes, it can be represented using $O(\sqrt{n}\lg n)$ bits such that the equivalence query and merge queries can be supported in $O(\lg n/\lg\lg n)$ worst case time. In fact, using the same space, equivalence query can be supported in $O(\alpha(n))$ amortized time and merge queries can be supported in $O(\alpha(n))$ expected amortized time, where $\alpha(n)$ is the inverse Ackermann function.*

Proof. (**sketch**) The primary structure we maintain is the one as in the proof of Theorem 5. To support merge operations, we maintain an auxiliary structure that captures the merges that have happened until $O(\sqrt{n})$ sets have merged. During this time, the original labeling of the elements is maintained. After $O(\sqrt{n})$ merges have happened, the entire data structure is reconstructed with relabeling of the elements using standard tricks and analysis.

The auxiliary structure needs to support insert, membership and union-find queries. By using a fusion tree [13,3] for the insert-membership structure and the union-find data structure of [1,6,22] we achieve the worst case bounds. By using a dynamic perfect hashing scheme [10] for insert-membership, the amortized bounds of the theorem follow.

Due to space limitations, we omit the details of the proof. □

We remark that as we change the labels of the original elements in our structure, the time and space bounds are better than related space efficient structures developed before (for example [5]). We discuss this issue in Conclusions.

5 Conclusions

We have discussed time-space tradeoffs for the fundamential problem of supporting equivalence queries. Our first result is an establishment of a tight bound for the label space required for the elements to answer equivalence query by just looking at the labels.

Then we showed that one can represent an equivalence relation on n elements using $O(\sqrt{n})$ bits of space, which is a constant factor of the information theoretically optimum number of bits required. Our scheme allows an implicit labeling of elements and supports equivalence queries in $O(\lg n)$ time. Improving this to constant time is an interesting open problem, though we could achieve constant time using $O(\sqrt{n}\log n)$ bits.

We also developed a space efficient dynamic structure where the merge operation can also be supported as fast as the standard (non-space-efficient) union-find structures using $O(\sqrt{n}\lg n)$ bits. Our main contribution is on the time-space tradeoffs for representing equivalence queries using clever use of several known structures.

Not withstanding our claim in Theorem 6 that we can support unions and finds on an n element set using $O(\sqrt{n}\lg n)$ bits and at the same asymptotic time as the best known (not so space efficient) structures, we don't know of a direct way to apply them for supporting static or (incremental) dynamic connectivity queries on graphs. This is because the original labels of the elements are modified to obtain our space efficient structure. Hence to support the user queries, we need to store the permutation that maps the user labels to our labels or update the user of the labelling (in the latter case, the user herself can answer the connectivity query). This is particularly important in our dynamic structure that supports union, as every so often, the labels are recomputed.

An example setting where our structures can be applicable is as follows. Consider a distributed environment where multiple processors are performing some intensive computation. These processors are space constrained, and each processor receives a request with a label from two different processors to perform some computation. Assume that the application requires the processor receiving this request to first determine whether the two labels are in the same equivalence class to perform the computation. So it does that using our small space union-find structure. In the case of dynamic (merge) queries, all processors must be aware of all the merges happening at any of the processors to perform their own local computation in case the labels get changed. Alternatively we can assume synchrony and communicate the relabelling after every change.

One useful query in this scenario that can be supported easily by our structure is the following. Each of the processors may have a small subset of labels about which they are particularly interested in. When a request for an equivalence query comes in, the processor may also want to know whether there is any element in its interest set that is in the same equivalence class as the query element. This can be supported in constant time by maintaining a succinct membership structure (as in [8]) for the classes in which the elements in its interest set belongs to, in addition to our succinct union-find structure.

Finally, given that union-find is a fundamental structure for representing equivalence classes, we feel that our structure and approach will find applications in other scenarios we haven't imagined.

Acknowledgement. The second author gratefully acknowledges the discussions he had with Tetsuo Asano which initiated work on the problem.

References

1. Alstrup, S., Ben-Amram, A.M., Rauhe, T.: Worst-case and amortised optimality in union-find. In: Proceedings of the 31st ACM Symposium on Theory of Computing (STOC), pp. 499–506 (1999)
2. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. SIAM Journal on Discrete Math. 19(2), 448–462 (2005)
3. Andersson, A., Miltersen, P.B., Thorup, M.: Fusion trees can be implemented with AC0 instructions only. Theoretical Computer Science 215, 337–344 (1999)
4. Barbay, J., Castelli Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. Algorithmica 62(1-2), 224–257 (2012)
5. Barbay, J., Gagie, T., Navarro, G., Nekrich, Y.: Alphabet partitioning for compressed rank/select and applications. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 315–326. Springer, Heidelberg (2010)
6. Blum, N.: On the single-operation worst-case time complexity of the disjoint set union problem. SIAM Journal on Computing 15(4), 1021–1024 (1986)
7. Blandford, D.K., Blelloch, G.E.: Compact representations of ordered sets. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 11–19 (2004)
8. Brodnik, A., Munro, J.I.: Membership in constant time and almost-minimum space. SIAM Journal on Computing 5, 1627–1640 (1999)
9. Delpratt, O., Rahman, N., Raman, R.: Compressed prefix sums. SOFSEM (1), 235–247 (2007)

10. Dietzfelbinger, M., Karlin, A., Mehlhorn, K., Meyer auf der Heide, F., Rohnert, H., Tarjan, R.E.: Dynamic perfect hashing: upper and lower bounds. SIAM Journal on Computing 23 (4), 738–761 (1994)
11. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graph. In: Proceedings of the European Symposium on Algorithms, pp. 393–404 (2008)
12. Franceschini, G., Muthukrishnan, S.M., Pătraşcu, M.: Radix sorting with no extra space. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 194–205. Springer, Heidelberg (2007)
13. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. 47(3), 424–436 (1993)
14. Grossi, R., Orlandi, A., Raman, R., Srinivasa Rao, S.: More haste, less waste: lowering the redundancy in fully indexable dictionaries. In: Proceedings of the 26th international symposium on Theoretical Aspects of Computer Science, STACS, pp. 517–528 (2009)
15. Gupta, A., Hon, W.-K., Shah, R., Vitter, J.S.: Compressed data structures: dictionaries and data-aware measures. Theoretical Computer Science 387(3), 313–331 (2007)
16. Hagerup, T.: Sorting and searching on the word RAM. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 366–398. Springer, Heidelberg (1998)
17. Hardy, G.H., Ramanujan, S.: Asymptotic formulae in combinatory analysis. Proceedings of the London Mathematical Society 17, 75–115 (1918)
18. Katz, M., Katz, N.A., Korman, A., Peleg, D.: Labeling schemes for flow and connectivity. SIAM Journal on Computing 34(1), 23–40 (2004)
19. Munro, J.I., Nicholson, P.K.: Succinct posets. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 743–754. Springer, Heidelberg (2012)
20. Raman, R.: Generating random graphs efficiently. In: Dehne, F., Fiala, F., Koczkodaj, W.W. (eds.) ICCI 1991. LNCS, vol. 497, pp. 149–160. Springer, Heidelberg (1991)
21. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees, multisets and prefix sums. ACM Transactions on Algorithms 3, 25 (2007)
22. Smid, M.H.M.: A data structure for the union-find problem having good single-operation complexity, ALCOM: Algorithms Review, Newsletter of the ESPRIT II Basic Research Actions Program (1990)
23. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. Information Processing Letters 17 (2), 81–84 (1983)

Sliding Bloom Filters[*]

Moni Naor[**] and Eylon Yogev

Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot 76100, Israel
{eylon.yogev,moni.naor}@weizmann.ac.il

Abstract. A Bloom filter is a method for reducing the space (memory) required for representing a set by allowing a small error probability. In this paper we consider a Sliding Bloom Filter: a data structure that, given a stream of elements, supports membership queries of the set of the last n elements (a sliding window), while allowing a small error probability and a slackness parameter. The problem of sliding Bloom filters has appeared in the literature in several communities, but this work is the first theoretical investigation of it.

We formally define the data structure and its relevant parameters and analyze the time and memory requirements needed to achieve them. We give a low space construction that runs in $O(1)$ time per update with high probability (that is, for all sequences with high probability all operations take constant time) and provide an almost matching lower bound on the space that shows that our construction has the best possible space consumption up to an additive lower order term.

1 Introduction

Given a stream of elements, we consider the task of determining whether an element has appeared in the last n elements of the stream. To accomplish this task, one must maintain a representation of the last n elements at each step. One issue, is that the memory required to represent them might be too large and hence an approximation is used. We formally define this approximation and completely characterize the space and time complexity needed for the task.

In 1970 Bloom [Blo70] suggested an efficient data structure, known as the '*Bloom filter*', for reducing the space required for representing a set S by allowing a small error probability on membership queries. The problem is also known as the approximate membership problem (however, we refer to any solution simply as a 'Bloom filter'). A solution is allowed an error probability of ε for elements not in S (false positives), but no errors for members of S. In this paper, we consider the task of efficiently maintaining a Bloom filter of the last n elements (called 'the sliding window') of a stream of elements.

We define an (n, m, ε)-*Sliding Bloom Filter* as the task of maintaining a Bloom filter over the last n elements. The answer on these elements must always be 'Yes',

[*] Research supported in part by a grant from the I-CORE Program of the Planning and Budgeting Committee, the Israel Science Foundation and the Citi Foundation.

[**] Incumbent of the Judith Kleeman Professorial Chair.

the m elements that appear prior to them have no restrictions (m is a slackness parameter) and for any other element the answers must be 'Yes' with probability at most ε. In case m is infinite, all elements prior to the current window have no restrictions, and we write for short (n, ε)-Sliding Bloom Filter.

The problem was studied in several communities and various solutions were suggested. In this work, we focus on a theoretical analysis of the problem and provide a rigorous analysis of the space and time needed for solving the task. We construct a Sliding Bloom Filter with $O(1)$ query and update time, where the running time is worst case with high probability (see the theorems in Section 1.2 for precise definitions) and has near optimal space consumption. We prove a matching space lower bound that is tight with our construction up to an additive lower order term. Roughly speaking, our main result is figuring out the first two terms of the space required by a Sliding Bloom Filter: $n \log \frac{1}{\varepsilon} + n \cdot \max\left\{\log\log \frac{1}{\varepsilon}, \log \frac{n}{m}\right\}$

A simple solution to the task is to partition the window into blocks of size m and for each block maintain its own Bloom filter. This results in maintaining $\left\lceil \frac{n}{m} + 1 \right\rceil$ Bloom filters. To determine if an element appeared or not we query all the Bloom filters and answer 'Yes' if any of them answered positively. There are immediate drawbacks of this solution, even assuming the Bloom filters are optimal in space and time:

- Slow query time: $\left\lceil \frac{n}{m} + 1 \right\rceil$ Bloom filter lookups.
- High error probability: since an error can occur on each block, to achieve an effective error probability of ε we need to set each Bloom filter to have error $\varepsilon' = \frac{\varepsilon m}{n+m}$, which means that the total space used has to grow (relative to a simple Bloom filter) by roughly $n \log \frac{n+m}{m}$ bits (see Section 1.3).
- Sub-optimal space consumption for large m: the first two drawbacks are acute for small m, but when m is large, say $n = m$, then each block is large which results in a large portion of the memory being 'wasted' on old elements.

We overcome all of the above drawbacks: the query time is always constant and for *any* m the space consumption is nearly optimal.

Sliding Bloom Filters can be used in a wide range of applications and we discuss two settings where they are applicable and have been suggested. In one setting, Bloom filters are used to quickly determine whether an element is in a local web cache [FCAB00], instead of querying the cache which may be slow. Since the cache has limited size, it usually stores the least recently used items (LRU policy). A Sliding Bloom Filter is used to represent the last n elements used and thus, maintain a representation of the cache's contents at any point in time.

Another setting consists of the task of identifying duplicates in streams. In many cases, we consider the stream to be unbounded, which makes it impractical to store the entire data set and answer queries precisely and quickly. Instead, it may suffice to find duplicates over a sliding window while allowing some errors. In this case, a Sliding Bloom Filter (with m set to infinity) suffices and in fact, we completely characterize the space complexity needed for this problem.

1.1 Problem Definition

Given a stream of elements $\sigma = x_1, x_2, ...$ from a finite universe U of size u, parameters n, m and ε, such that $n < \varepsilon u$, we want to approximately represent a sliding window of the n most recent elements of the stream. An algorithm A is given the elements of the stream one by one, and does not have access to previous elements that were not stored explicitly. Let $\sigma_t = x_1, \ldots, x_t$ be the first t elements of the stream σ and let $\sigma_t(k) = x_{\max(0,t-k+1)}, \ldots, x_t$ be the last k elements of the stream σ_t. At any step t the current window is $\sigma_t(n)$ and the m elements before them are $\sigma_{t-n}(m)$. If $m = \infty$ then define $\sigma_{t-n}(m) = x_1, \ldots, x_{t-n}$. Denote $A(\sigma_t, x) \in \{\text{'Yes'}, \text{'No'}\}$ the result of the algorithm on input x given the stream σ_t. We call A an (n, m, ε)-*Sliding Bloom Filter* if for any $t \geq 1$ the following two conditions hold:

1. For any $x \in \sigma_t(n)$: $\Pr[A(x) = \text{'Yes'}] = 1$
2. For any $x \notin \sigma_t(n+m)$: $\Pr[A(x) = \text{'Yes'}] \leq \varepsilon$

where the probability is taken over the internal randomness of the algorithm A. Notice that for an element $x \in \sigma_{t-n}(m)$ the algorithm may answer arbitrarily (no restrictions). See Figure 1.

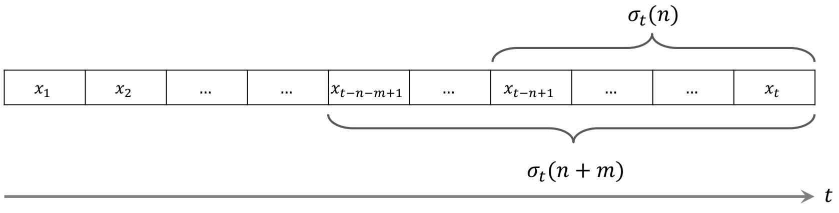


Fig. 1. The sliding window of the last n and $n + m$ elements

An algorithm A for solving the problem is measured by its memory consumption, the time it takes to process each element and answer a query. We denote by $|A|$ the maximum number of bits used by A at any step. The model we consider is the unit cost RAM model in which the elements are taken from a universe of size u, and each element can be stored in a single word of length $w = \log u$ bits. Any operation in the standard instruction set can be executed in constant time on w-bit operands. This includes addition, subtraction, bitwise Boolean operations, left and right bit shifts by an arbitrarily number of positions, and multiplication. The unit cost RAM model is considered the standard model for the analysis of the efficiency of data structures.

According to the above definition we are not assured that there are no 'bad' points in time where the data structure accepts many false positives. We call the desired property that throughout the lifetime of the data structure at *all* points in the time the number of elements in the universe on which a false positive is returned to be $\varepsilon \cdot u$ the *absolute false positive assumption.* The absolute false positive assumption is a very desirable property from a Sliding Bloom Filter and reasonable constructions enjoy it. We use it in the proof in Section 3, and in the full version [NY13] we show how to get the same lower bound without it. An example of a Sliding Bloom Filter for which the assumption does not hold can be obtained by taking any (n, ε)-Sliding Bloom Filter and modifying it such that it chooses a random index $k \in [1, n]$ and at step k of the stream it always answers 'Yes'. This results in an $(n, \varepsilon + \frac{1}{n})$-Sliding Bloom Filter in which there will *always* be some step at which the false positive rate is high (it is 1).

1.2 Our Contributions

We provide tight upper and lower bounds to the (n, m, ε)-problem. In fact, we achieve space optimality up to the second term. Our first contribution is a construction of an efficient Sliding Bloom Filter: it has query time $O(1)$ worst case and update time $O(1)$ worst case with high probability, for the entire sequence. For $\varepsilon = o(1)$ the space consumption is near optimal: the two leading terms are optimal in constants.

Theorem 1. *For any $m > 0$, and sufficiently large n there exists an (n, m, ε)-Sliding Bloom Filter having the following space and time complexity on a unit cost RAM:*

Time*: Query time is $O(1)$ worst case. For any polynomial $p(n)$ and sequence of at most $p(n)$ operations, with probability at least $1 - 1/p(n)$, over the internal randomness of the data structure, all insertions are performed in time $O(1)$ worst case.*

Space*: the space consumption is: $(1+o(1))\left(n \log \frac{1}{\varepsilon} + n \cdot \max\left\{\log \frac{n}{m}, \log\log \frac{1}{\varepsilon}\right\}\right)$. In particular, for constant error ε we get that the space consumption is: $n \log\left(\frac{n}{m}\right) + O(n)$. Otherwise, for sub-constant ε that satisfies $\varepsilon = 2^{-O(\log^{1/3} n)}$ we get that:*

1. *If $m \geq \varepsilon n$ then the space consumption is: $n \log \frac{1}{\varepsilon} + n \cdot \max\left\{\log \frac{n}{m}, \log\log \frac{1}{\varepsilon}\right\} + O(n)$*
2. *If $m < \varepsilon n$ then the space consumption is: $n \log \frac{1}{\varepsilon} + (1 + o(1)) n \log \frac{n}{m}$*

The challenge we face is achieving constant time operations while space consumption remains very tight. In designing our algorithm we assemble ideas from several previous works along with new ones. The basic skeleton of the algorithm shares ideas with the work of Zhang and Guan [ZG08], however, their algorithm is based on the traditional Bloom filter and has immediate drawbacks: running time is super-constant and the space is far from optimal. To get an error probability of ε they use $M = O(n \log n \log \frac{1}{\varepsilon})$ bits, and moreover this is assuming the availability of truly random hash functions.

Thorup [Tho11] considered a similar data structure of hash tables with timeouts based on linear probing. He did not allow any error probability nor any slackness (i.e. $\varepsilon = 0$ and $m = 0$ in our terminology). The query time, as is general for linear probing, is only constant in expectation, and the space is only optimal within a constant factor.

Pagh, Pagh and Rao [PPR05] showed that the traditional construction of a Bloom filter can be replaced with a construction that is based on dictionaries. The dictionary based Bloom filter has the advantage that its running time and space consumption are completely determined by the dictionary itself, and it does not assume availability of truly random functions. Given the developments in succinct dictionaries, using this alternative has become more appealing.

Our algorithm is conceptually similar to the work of Zhang and Guan. However, we replace the traditional implementation of the Bloom filter with a dictionary based one. As the underlying dictionary, we use the state of the art dictionary given by Arbitman, Naor and Segev [ANS10], known as Backyard Cuckoo Hashing. Then we apply a similar method of lazy deletions as used by Thorup on the Backyard Cuckoo Hashing dictionary. Moreover, we introduce a slackness parameter m and instead of storing the exact index of each element we show a trade-off parameter c between the accuracy of the index stored and the number of elements we store in the dictionary. Optimizing c along with the combined methods described gives us the desired result: constant running time, space consumption of nearly $n \log \frac{1}{\varepsilon} + n \cdot \max \left\{ \log\log \frac{1}{\varepsilon}, \log \frac{n}{m} \right\}$ which is optimal in both leading constants and no assumption on the availability of truly random functions. We inherit the implementation complexity of the dictionary, and given an implementation of one, it is relatively simple to complete the algorithm's implementation.

Our second contribution, and technically the more involved one, is a matching space lower bound. We prove that if $\varepsilon = o(1)$ then any Sliding Bloom Filter must use space that is within an additive low order term of the space of our construction, regardless of its running time. For simplicity, we first assume that the Sliding Bloom Filter has the desired property that throughout the lifetime of the data structure at *all* points the number of false positives is not too large (recall that we denote this the *absolute false positive assumption*). In the full version [NY13] we prove the same result without this assumption.

Theorem 2. *Let A be an (n,m,ε)-Sliding Bloom Filter where $n < \varepsilon u$, then*

1. *If $m > 0$ then $|A| \geq n \log \frac{1}{\varepsilon} + n \cdot \max \left\{ \log \frac{n}{m}, \log\log \frac{1}{\varepsilon} \right\} - O(n)$*
2. *If $m = \infty$ then $|A| \geq n \log \frac{1}{\varepsilon} + n \log\log \frac{1}{\varepsilon} - O(n)$*

From Theorems 1 and 2 we conclude that making m larger than $n/\log \frac{1}{\varepsilon}$ does not make sense: one gets the same result for any value in $[n/\log \frac{1}{\varepsilon}, \infty)$. When m is small (less than εn), then the dominant expression in both the upper and lower bounds is $n \log \left(\frac{n}{m} \right)$.

The lower bound is proved by an encoding argument which is a common way of showing lower bounds in this area (see for example [PSW13]). Specifically, the idea of the proof is to use A to encode a set S and a permutation π on the set corresponding to the order of the elements in the set. We consider the number of steps from the point an element is inserted to A to the first point where A answers 'No' on it, and we define λ to be the sum of n such lengths. If λ is large, then there is a point where A represents a large portion of S, which benefits in the encoding of S. If λ is small, then A can be used as an approximation of

π, thus encoding π precisely requires a small amount of bits. In either case, the encoding must be larger than the entropy lower bound[1] which yields a bound on the size of A. The optimal value of the trade-off between representing a larger set or representing a more accurate ordering is achieved by our construction. In this sense, our upper bound and lower bound match not only by 'value' but also by 'structure'.

1.3 Related Work and Background

The data structure for the approximate set membership as suggested by Bloom in 1970 [Blo70] is relatively simple: it consists of a bit array which is initiated to '0' and k random hash functions. Each element is mapped to k locations in the bit array using the hash functions. To insert an element set all k locations to 1. On lookup return 'Yes' if all k locations are 1. To achieve an error probability of ε for a set of size n Bloom showed that if $k = \log \frac{1}{\varepsilon}$ then the length of the bit array should be roughly $1.44n \log \frac{1}{\varepsilon}$ (where the 1.44 is an approximation of $\log_2(e)$). Since its introduction Bloom filters have been investigated extensively and many variants, implementations and applications have been suggested. We call any data structure that implements the approximate set membership a 'Bloom filter'. A comprehensive survey (for its time) is Broder and Mitzenmacher [BM02].

A lot of attention was devoted for determining the exact space and time requirements of the approximate set membership problem. Carter et al. [CFG$^+$78] proved an entropy lower bound of $n \log \frac{1}{\varepsilon}$, when the universe U is large. They also provided a reduction from approximate membership to *exact* membership, which we use in our construction. The retrieval problem associates additional data with each element of the set. In the static setting, where the elements are fixed and given in advance, Dietzfelbinger and Pagh propose a reduction from the retrieval problem to approximate membership [DP08]. Their construction gets arbitrarily close to the entropy lower bound.

In the dynamic case, Lovett and Porat [LP10] proved that the entropy lower bound cannot be achieved for any *constant* error rate. They show a lower bound of $C(\varepsilon)\cdot n \log \frac{1}{\varepsilon}$ where $C(\varepsilon) > 1$ depends only on ε. Pagh, Segev and Wieder [PSW13] showed that if the size n is not known in advance then at least $(1 - o(1))n \log \frac{1}{\varepsilon} + \Omega(n \log \log n)$ bits of space must be used. The Sliding Bloom Filter is in particular also a Bloom Filter in a dynamic setting, thus the [LP10] and [PSW13] bounds are applicable.

As discussed, Pagh, Pagh and Rao [PPR05] suggested an alternative construction for the Bloom filter. They used the reduction of Carter et al. to improve the traditional Bloom filter in several ways: Lookup time becomes $O(1)$ independent of ε, has succinct space consumption, uses explicit hash functions and supports deletion. In the dynamic setting for a constant ε we do not know what is the leading term in the memory needed, however, for any sub-constant ε we know that the leading term is $n \log \frac{1}{\varepsilon}$: Arbitman, Naor and Segev present a solution,

[1] The entropy lower bound is base 2 logarithm of the size of the set of all possible inputs. In our case, all possible pairs (S, π).

called 'Backyard Cuckoo Hashing', which is optimal up to an additive lower order term (i.e., it is a succinct representation) [ANS10]. Thus, in this work we focus on sub-constant ε.

The model of sliding windows was first introduced by Datar et al. [DGIM02]. They consider maintaining an approximation of a statistic over a sliding window. They provide an efficient algorithm along with a matching lower bound.

Data structures for problems similar to the Sliding Bloom Filters have been studied in the literature quite extensively over the past years. The simple solution using $m = n$ consists of two large Bloom filters which are used alternatively. This method known as *double buffering* was proposed for classifying packets caches [CFL04]. Yoon [Yoo10] improved this method by using the two buffers simultaneously to increase the capacity of the data structure. Deng and Rafiei [DR06] introduced the Stable Bloom filter and used it to approximately detect duplicates in stream. Instead of a bit array they use an array of counters and to insert an element they set all associated counters to the maximal value. At each step, they randomly choose counters to decrease and hence older element have higher probability of being decreased and eventually evicted over time. Metwally et al. [MAA05] showed how to use Bloom filters to identify duplicates in click streams. They considered three models: Sliding Windows, Landmark Windows and Jumping Windows and discuss their relations. A comprehensive survey including many variations is given by Tarkoma et al. [TRL12]. However, as far as we can tell, no formal definition of a Sliding Bloom Filter as well as a rigorous analysis of its space and time complexity, appeared before.

2 The Construction of a Succinct Sliding Bloom Filter

Our algorithm uses a combination of transforming the approximate membership problem to the exact membership problem plus a solution to the retrieval problem. On an input x, we store $h(x)$, for some hash function h, in a dynamic dictionary and in addition store some information on the last time where x appeared. We consider the stream to be divided into generations of size n/c each, where c is a parameter is optimized at the end. At each step, we maintain a set S that represents the last $c+1$ generations and count the generations mod $(c+1)$. In addition to storing $h(x)$, we associate $s = \log(c+1)$ bits indicating the generation of x. Then, on query x we check if $h(x)$ is in the dictionary and that its associated generation is not more than $c+1$ generations old. Additionally, we need to delete old elements, before they interfere the other operations. The full details of the construction and proof of its correctness are in the full version of this paper [NY13].

3 A Tight Space Lower Bound

In this section we present a matching space lower bound to our construction. For simplicity, we use the false positive assumption and in the full version [NY13] we show how to get the same result without it.

Theorem 3. *Let A be an (n, m, ε)-Sliding Bloom Filter where $n < \varepsilon u$. If for any stream σ it holds that* $\Pr[\exists i \leq 3n : |\{x \in U : A(\sigma_i, x) = \text{`}Yes'\}| \geq n + 2\varepsilon u] \leq \frac{1}{2}$ *then*

1. *If $m > 0$ then $|A| \geq n \log \frac{1}{\varepsilon} + n \cdot \max\left\{\log \frac{n}{m}, \log\log \frac{1}{\varepsilon}\right\} - O(n)$*
2. *If $m = \infty$ then $|A| \geq n \log \frac{1}{\varepsilon} + n \log\log \frac{1}{\varepsilon} - O(n)$*

Proof. Let A be an algorithm satisfying the requirements in the statement of the theorem. The main idea of the proof is to use A to encode and decode a set $S \subset U$ and a permutation π on the set (i.e. an ordered set). Giving S to A as a stream, ordered by π, creates an encoding of an approximation of S and π: S is approximated by the set of all the elements for which A answers 'Yes' (denoted by $\mu_A(S)$), and π is approximated by the number of elements needed to be added to the stream in order for A to "release" each of the elements in S (that is, to answer 'No' on it). Then, to get an exact encoding, we encode only the elements of S from within the set $\mu_A(S)$. To get an exact encoding of π we encode only the difference between the location i of each element and the actual location it has been released. The key is to find the point where A best approximates S and π *simultaneously.*

Denote by A_r the algorithm with fixed random string r and let $\mu_{A_r}(\sigma) = \{x : A_r(\sigma, x) = \text{`Yes'}\}$. We show that w.l.o.g. we can consider A to be deterministic. Let $V = \{\sigma : |\sigma| = 2n\}$ be the set of all sequences of $2n$ *distinct* elements, and let $V(r) \subseteq V$ be the subset of inputs such that $|\mu_{A_r}(\sigma_i)| \leq n+2\varepsilon u$ for all $1 \leq i \leq 3n$. Since we assumed that for any σ we have that $\Pr_r[\exists i \leq 3n : |\mu_{A_r}(\sigma_i)| \geq n + 2\varepsilon u] \leq \frac{1}{2}$ then there must exist an r^* such that $|V(r^*)| \geq |V|/2$. Thus, we can assume that A is deterministic and encode only sequences from $V(r^*)$. Then the encoding lower bound changes from $\log|V|$ to $\log(|V|/2) = \log|V| - 1$. This loss of 1 bit is captured by the lower order term $O(n)$ in the lower bound, and hence can be ignored.

Notice that r^* need not be explicitly specified in the encoding since the decoder can compute it using the description of the algorithm A (which may be part of its fixed program). From now on, we assume that A is deterministic (and remove the A_r notation) and assume that for any $\sigma \in V(r^*)$ we have that $\mu_A(\sigma) \leq n + 2\varepsilon u \leq 3\varepsilon u$.

We now make an important definition:

$$\ell(\sigma, x) = \min\{\arg\min_k \{\exists y_1, \ldots, y_k \in U : A(\sigma y_1 \cdots y_k, x) = 0\}, n, m\}$$

$\ell(\sigma, x)$ is the minimum number of elements needed to be added to σ such that A answers 'No' on x. Notice that $\ell(\sigma, \cdot)$ can be computed for any set S given the representation of $A(\sigma)$.

We encode any set S of size $2n$ and a permutation $\pi : [2n] \to [2n]$ using A. After encoding S we compare the encoding length to the entropy lower bound of $\mathcal{B}(u, 2n) + \log((2n)!)$. Consider applying π on (some canonical order of) the elements of S and let $x_1, \ldots, x_{2n}$ be the resulting elements of S ordered by π. For any $i > 2n$ let $x_i = x_{i-2n}$, then for any $k \geq 1$ define the sequence $\sigma_k = x_1, \ldots, x_k$. Let $\phi(\sigma_k) = \mu(\sigma_k) \cap S$ and define $\Delta(\sigma_k, i) = \ell(\sigma_k, x_i) + (k - n) - i$. Notice that, given $A(\sigma_k)$, $\Delta(\sigma_k, i)$, k and n one can compute the position i of the element x_i. Define

$$\lambda_k = \sum_{i=k-n+1}^{k} \Delta(\sigma_k, i), \text{ and } \lambda = \max_{n \le k \le n} \lambda_k$$

If $m \ge n$ (or $m = \infty$) then $0 \le \lambda \le n^2$, otherwise $0 \le \lambda \le nm$

Lemma 1. *Let* $k \in [n, 2n]$ *then* $\sum_{j=k}^{k+n-1} |\phi(\sigma_j)| \ge n^2 + \lambda_k$.

Proof. Instead of summing over $\phi(\sigma_j)$, we sum over x_i and count the number of $\phi(\sigma_j)$ such that $x_i \in \phi(\sigma_j)$. For $k - n + 1 \le i \le k$ we know that $x_i \in \sigma_k(n)$ and by the definition of $\ell(\sigma_k, x_i)$ we get that $x_i \in \phi(\sigma_k), \dots, \phi(\sigma_{k+\ell(\sigma_k,x_i)-1})$. For $k + 1 \le i \le k + n - 1$ we know that $x_i \in \phi(\sigma_i), \dots, \phi(\sigma_{k+n-1})$. Therefore:

$$\begin{aligned}\sum_{j=k}^{k+n-1} |\phi(\sigma_j)| &\ge \sum_{i=k-n+1}^{k} \ell(\sigma_k, x_i) + \sum_{i=k+1}^{k+n-1} (k + n - i) \\ &= \sum_{i=k-n+1}^{k} \Delta(\sigma_k, i) + n^2 = \lambda_k + n^2\end{aligned}$$

By averaging, we get that for any k there exist some $j \in [k, k+n-1]$ such that $|\phi(\sigma_j)| \ge n + \frac{\lambda_j}{n}$. Let k^* be such that $\lambda = \lambda_{k^*}$, then we know that there exist some $j^* \in [k^*, k^*+n-1]$ such that $|\phi(\sigma_j)| \ge n + \frac{\lambda}{n}$. Note that $n \le j^* \le k^* + n - 1 \le 3n$ which is in the range of indices of the false positive assumption.

We include the memory representation of $A(\sigma_{j^*})$ in the encoding. The decoder uses this to compute the set $\mu(\sigma_{j^*})$, which by the absolute false positive definition we know that $|\mu(\sigma_{j^*})| \le 3\varepsilon u$. Since $|\phi(\sigma_{j^*})| \ge n + \frac{\lambda}{n}$, we need only $\mathcal{B}(3\varepsilon u, n + \frac{\lambda}{n})$ bits to encode $n + \frac{\lambda}{n}$ elements of S out of them. The remaining $n - \frac{\lambda}{n}$ elements are encoded explicitly using $\mathcal{B}(u, n - \frac{\lambda}{n})$ bits. This completes the encoding of S.

To encode π we need the decoder to be able to extract i for each x_i. For any $x_i \in \sigma_{j^*}(n)$ the decoder uses $A(\sigma_{j^*})$ and computes $\ell(\sigma_{j^*}, x_i)$. Now, in order for the decoder to exactly decode i we need to encode all the $\Delta(\sigma_{j^*,i})$'s. Since $\sum_{i=j^*-n+1}^{j^*} \Delta(\sigma_{j^*,i}) = \lambda_{j^*} \le \lambda$ we can encode all the $\Delta(\sigma_{j^*,i})$'s using $\log \binom{n+\lambda}{n}$ bits (balls and sticks method), and the remaining elements' positions will be explicitly encoded using $n \log n$ bits. Denote by $|A|$ the number of bits used by A. Comparing the encoding length to the entropy lower bound we get

$$|A| + \log \binom{3\varepsilon u}{n + \frac{\lambda}{n}} + \log \binom{u}{n - \frac{\lambda}{n}} + \log \binom{\lambda + n}{n} + n \log n \ge \log \binom{u}{2n} + \log((2n)!)$$

and therefore

$$|A| \ge (n + \frac{\lambda}{n}) \log \frac{1}{\varepsilon} + (n + \frac{\lambda}{n}) \log n + (n - \frac{\lambda}{n}) \log (n - \frac{\lambda}{n}) - n \log (\lambda + n) - O(n)$$

Consider two possible cases for λ. If $\lambda \leq 0.9n^2$ then we get

$$|A| \geq (n + \frac{\lambda}{n}) \log \frac{1}{\varepsilon} + 2n \log n - n \log(\lambda + n) - O(n)$$

The minimum of this expression, as a function of λ, is achieved at $\lambda = \frac{n^2}{\log \frac{1}{\varepsilon}} - n$. If $m \geq \frac{n}{\log \frac{1}{\varepsilon}} - 1$ then the minimum can be achieved and we get that

$$|A| \geq n \log \frac{1}{\varepsilon} + n \log\log \frac{1}{\varepsilon} - O(n).$$

Otherwise, if $m < \frac{n}{\log \frac{1}{\varepsilon}} - 1$ then $\lambda \leq mn \leq \frac{n^2}{\log \frac{1}{\varepsilon}} - n$ and minimum value will be achieved at $\lambda = nm$ which yields the required lower bound:

$$|A| \geq n \log \frac{1}{\varepsilon} + n \log \frac{n}{m} - O(n).$$

If $0.9n^2 < \lambda \leq n^2$ then $m \geq 0.9n \geq \frac{n}{\log \frac{1}{\varepsilon}}$. Thus, we get that

$$|A| \geq (n + \frac{\lambda}{n}) \log \frac{1}{\varepsilon} - (n - \frac{\lambda}{n}) \log n + (n - \frac{\lambda}{n}) \log(n - \frac{\lambda}{n}) - O(n)$$

the minimum of this expression, as a function of λ between the given range is achieved at $\lambda = 0.9n^2$ which yields the desired bound:

$$|A| \geq n \log \frac{1}{\varepsilon} + n \log\log \frac{1}{\varepsilon} - O(n).$$

Acknowledgments. We thank Ilan Komargodski, Tal Wagner and the anonymous referees for many useful comments.

References

[ANS10] Arbitman, Y., Naor, M., Segev, G.: Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In: 2010 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 787–796. IEEE (2010)

[Blo70] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)

[BM02] Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. In: Internet Mathematics, pp. 636–646 (2002)

[CFG$^+$78] Carter, L., Floyd, R., Gill, J., Markowsky, G., Wegman, M.: Exact and approximate membership testers. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, pp. 59–65. ACM (1978)

[CFL04] Chang, F., Feng, W.-C., Li, K.: Approximate caches for packet classification. In: Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004, vol. 4, pp. 2196–2207. IEEE (2004)

[DGIM02] Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM Journal on Computing 31(6), 1794–1813 (2002)

[DP08] Dietzfelbinger, M., Pagh, R.: Succinct data structures for retrieval and approximate membership. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 385–396. Springer, Heidelberg (2008)

[DR06] Deng, F., Rafiei, D.: Approximately detecting duplicates for streaming data using stable bloom filters. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 25–36. ACM (2006)

[FCAB00] Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. 8(3), 281–293 (2000)

[LP10] Lovett, S., Porat, E.: A lower bound for dynamic approximate membership data structures. In: 2010 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 797–804. IEEE (2010)

[MAA05] Metwally, A., Agrawal, D., Abbadi, A.E.: Duplicate detection in click streams. In: Proceedings of the 14th International Conference on World Wide Web, pp. 12–21. ACM (2005)

[NY13] Naor, M., Yogev, E.: Sliding bloom filters. CoRR, abs/1304.5872 (2013)

[PPR05] Pagh, A., Pagh, R., Srinivasa Rao, S.: An optimal bloom filter replacement. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 823–829. Society for Industrial and Applied Mathematics (2005)

[PSW13] Pagh, R., Segev, G., Wieder, U.: How to approximate a set without knowing its size in advance. arXiv preprint arXiv:1304.1188 (2013)

[Tho11] Thorup, M.: Timeouts with time-reversed linear probing. In: INFOCOM, pp. 166–170 (2011)

[TRL12] Tarkoma, S., Rothenberg, C.E., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. IEEE Communications Surveys & Tutorials 14(1), 131–155 (2012)

[Yoo10] Yoon, M.K.: Aging bloom filter with two active buffers for dynamic sets. IEEE Transactions on Knowledge and Data Engineering 22(1), 134–138 (2010)

[ZG08] Zhang, L., Guan, Y.: Detecting click fraud in pay-per-click streams of online advertising networks. In: ICDCS, pp. 77–84 (2008)

Vertex-Weighted Matching in Two-Directional Orthogonal Ray Graphs

C. Gregory Plaxton[⋆]

Department of Computer Science
University of Texas at Austin
plaxton@cs.utexas.edu

Abstract. Let G denote an n-vertex two-directional orthogonal ray graph. A bicolored 2D representation of G requires only $O(n)$ space, regardless of the number of edges in G. Given such a compact representation of G, and a (possibly negative) weight for each vertex, we show how to compute a maximum weight matching of G in $O(n \log^2 n)$ time. The classic problem of scheduling weighted unit tasks with release times and deadlines is a special case of this problem, and we obtain an $O(n \log n)$ time bound for this special case. As an application of our more general result, we obtain an $O(n \log^2 n)$-time algorithm for computing the VCG outcome of a sealed-bid unit-demand auction in which each item has two associated numerical parameters (e.g., third-party "quality" and "seller reliability" scores) and each bid specifies the amount an agent is willing to pay for any item meeting specified lower bound constraints with respect to these two parameters.

1 Introduction

Certain natural classes of graphs can be represented using a constant number of words of storage for each vertex, and no additional storage for each edge, since the edges are represented implicitly: Given the representation of two vertices u and v, it is possible to determine in constant time whether there is an edge between u and v. For example, consider the class of bipartite graphs where each "left vertex" corresponds to a unit job (i.e., a job requiring one unit of processing time) with a specified integer release time and deadline, each "right vertex" corresponds to a unit-time slot on a shared resource with a specified integer timestamp, and there is an edge between left vertex u and right vertex v if and only if the timestamp of the slot associated with v lies in the interval specified by the release time and deadline of the job associated with u. Such a bipartite graph is said to be "convex".

For such "compactly representable" classes of graphs, it is interesting to revisit the complexity of fundamental graph problems. By working directly with

[⋆] Department of Computer Science, University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, Texas 78712–1757. This research was supported by NSF Grant CCF–1217980.

the compact representation, we seek to outperform traditional algorithms designed for the standard adjacency list representation. In this paper, we revisit the complexity of vertex-weighted matching problems on certain compactly representable classes of bipartite graphs. All of the algorithms that we develop have time complexity that is quasilinear (i.e., within a polylogarithmic factor of linear) in the number of vertices.

Matching algorithms for convex bipartite graphs have received significant attention in the literature. In the following discussion, U denotes the set of left vertices, and V denotes the set of right vertices, of a given convex bipartite graph. Glover presented a simple greedy algorithm [8] for maximum-cardinality convex bipartite matching that admits an $O(|V| + |U| \log |U|)$-time implementation using an elementary priority queue data structure. Later, van Emde Boas used a fast priority queue to obtain an $O(|V| + |U| \log \log |U|)$-time implementation of Glover's algorithm [21]. Lipski and Preparata [10] used Tarjan's fast union-find data structure [20] to devise a different algorithm running in time $O(|U| + |V|\alpha(|V|))$, where α is a functional inverse of Ackermann's algorithm. Gabow and Tarjan [6] show that this application of union-find falls into a category admitting a linear-time implementation, thereby reducing the Lipski-Preparata time bound to $O(|U| + |V|)$. Another line of work focused on eliminating the dependence of the running time on $|V|$ [7,15], at the expense of introducing a mild technical assumption regarding the input representation. This research culminated in the $O(|U|)$-time algorithm of Steiner and Yeomans [19].

In terms of vertex-weighted matching algorithms for convex bipartite graphs, most prior research has focused on the "left-weighted" special case in which all of the right vertices have zero weight, which corresponds to the classic problem of scheduling weighted unit jobs with release times and deadlines. (In the notation of Section 2, this corresponds to the LMWM and LMWMCM problems, which are essentially equivalent.) Dekel and Sahni [5] present a parallel algorithm for left-weighted convex bipartite matching that uses $O(|U|^2)$ processors and $O(\log^2 |U|)$ time, and which is based on a sequential algorithm with $O(|U|^2)$ complexity. Brodal et al. [2] present a data structure based on the Dekel-Sahni algorithm for the problem of maintaining a maximum cardinality matching in a dynamic convex bipartite graph. Lipski and Preparata [10] use the matroid greedy framework to develop a left-weighted convex bipartite matching algorithm with time complexity $O(|U|^2 + |U| \cdot |V|)$. Plaxton [13] discusses a similar algorithm based on the matroid greedy framework, and shows how to implement this algorithm in $O(|U| \log |U| + |V| \log^2 |V|)$ time using a data structure based on augmented trees. With additional preprocessing, and making the same technical assumption regarding the input representation as in Steiner and Yeomans [19], Plaxton improves this bound to $O(|U| + k \log^2 k)$, where $k \leq \min\{|U|, |V|\}$ denotes the size of a maximum cardinality matching.

Katriel [9] presents an $O(|E| + |V| \log |U|)$-time algorithm for the right-weighted special case of vertex-weighted matching in convex bipartite graphs. (Here E denotes the edge set of the graph.) Katriel obtains the same time bound algorithm for the general vertex-weighted matching problem in convex bipartite graphs, under the restriction that the input graph $G = (U, V, E)$ admits a matching of size $|U|$. Since the input size is $\Theta(|U| + |V|)$, and $|E|$ could be as large as $\Theta(|U| \cdot |V|)$, these algorithms have quadratic complexity.

In this paper, we present quasilinear vertex-weighted matching algorithms for a class of bipartite graphs that properly contains the class of convex bipartite graphs. The bipartite graphs that we study admit a representation in which each vertex (left or right) has an x-value and a y-value, and there is an edge from a left vertex u to a right vertex v if and only if the x-value of u is at most the x-value of v and the y-value of u is at most the y-value of v. Such a bipartite graph is called a two-dimensional orthogonal ray graph, or 2DORG (see Section 4 for a more formal definition). It is easy to see how to represent a convex bipartite graph as a 2DORG: Using the job-slot terminology introduced earlier, we can set the x-value (resp., y-value) of each left vertex to the release time (resp., negation of the deadline) of the associated job, and we can set the x-value (resp., y-value) of each right vertex to the timestamp (resp., negation of the timestamp) of the associated slot. On the other hand, the class of 2DORGs is substantially richer than the class of convex bipartite graphs. For example, it is known that the class of 2DORGs properly contains the class of interval bigraphs, which in turn properly contains the class of convex bipartite graphs [16].

We now discuss the key techniques underlying our results. A starting point for our work is the elegant linear-time algorithm of Chang [3] for computing a maximum cardinality matching (MCM) of a chordal bipartite graph. The class of chordal bipartite graphs properly contains the class of 2DORGs [16]. Chang's algorithm runs in $O(m + n)$ time on an input graph with m edges and n vertices. Recall that in the present work we are seeking running times that are quasilinear in n. We obtain an $O(n \log n)$-time implementation of Chang's algorithm by making use of a suitably augmented binary search tree (BST). Our augmented BST data structure may be viewed as a special case of the priority search tree data structure of McCreight [11]. (For a good introduction to the topic of augmented BST data structures, see Cormen et al. [4, Chapter 14].)

Most of the technical work in our paper is geared towards leveraging the aforementioned $O(n \log n)$-time MCM algorithm for 2DORGs to obtain an $O(n \log^2 n)$-time vertex-weighted matching algorithms for 2DORGs. Here we exploit the vertex-weighted matching framework of Spencer and Mayr [18]. This is a divide-and-conquer framework for reducing vertex-weighted matching to unweighted matching. The framework is valid for general (bipartite or nonbipartite) graphs. As in the case of Chang's algorithm discussed in the previous paragraph, the original Spencer-Mayr framework is not geared towards obtaining running times that are quasilinear in the number of vertices. Rather, the original framework seeks fast running times for general graphs; these bounds depend on m and n and are not quasilinear in n, even for sparse graphs. We identify a small number of basic primitives that suffice to support the Spencer-Mayr framework, and show how to implement each of these primitives in $O(n \log n)$ time on 2DORGs. One such primitive is the $O(n \log n)$-time MCM algorithm discussed in the previous paragraph. Given a current matching, another key primitive identifies all of the vertices that can be reached from some unmatched left vertex via an alternating path of unmatched and matched edges. As in the case of our MCM algorithm for 2DORGs, our $O(n \log n)$-time 2DORG implementation of the latter primitive is based on augmented BSTs. Once we establish that all of the primitives associated with the Spencer-Mayr framework admit $O(n \log n)$-time implementations on 2DORGs, we find that the resulting divide-and-conquer recurrence solves to give an overall running time of $O(n \log^2 n)$ for vertex-weighted matching in 2DORGs.

A practical motivation for the work of the present paper is to better understand the class of sealed-bid unit-demand auctions for which it is possible to compute a suitable outcome in time that is quasilinear in the number of vertices. In certain real-time applications of combinatorial auctions, it is crucial to employ mechanisms with low time complexity. For example, in the realm of sponsored search auctions, each search query triggers a combinatorial auction in which a (potentially large) number of bidders vie for a collection of ad slots; such an auction needs to be resolved rapidly so that the search results can be provided in a timely manner. We use our vertex-weighted matching algorithm for 2DORGs to compute a VCG allocation for a certain class of sealed-bid unit-demand auctions in $O(n \log^2 n)$ time, and we show how to compute the VCG prices in $O(n \log n)$ additional time.

The remainder of the paper is organized as follows. Section 2 provides some basic definitions and lemmas. Section 3 introduces ordered and elimination-ordered representation schemes. Section 4 presents our main result, an $O(n \log^2 n)$-time algorithm to compute a maximum weight matching of any n-vertex 2DORG. Due to space limitations, some details are omitted from this conference version. The companion technical report [14] includes all of the material in the present version plus four appendices. Appendix A of [14] reviews the relevant aspects of the Spencer-Mayr vertex-weighted matching framework, and adapts this framework to our setting. Appendix B of [14] presents an $O(n \log n)$-time algorithm for the special case of left-weighted matching on convex bipartite graphs. Appendix C of [14] presents several useful lemmas. Appendix D of [14] describes an $O(n \log^2 n)$-time algorithm for computing the VCG outcome of a 2DORG-related class of sealed-bid unit-demand auctions.

2 Preliminaries

A *matching* of a graph $G = (V, E)$ is a subset E' of E such that the $2|E'|$ endpoints of the edges in E' are all distinct. A *maximum cardinality matching (MCM)* of G is a matching M of G such that $|M| \geq |M'|$ for all matchings M' of G. If each edge of G has an associated weight, we define the weight of a matching M, denoted $w(M)$, as the sum of the weights of its associated edges. A *maximum weight matching (MWM)* of G is a matching M of G such that $w(M) \geq w(M')$ for all matchings M' of G. A *maximum weight MCM (MWMCM)* of G is an MCM M of G such that $w(M) \geq w(M')$ for all MCMs M' of G.

This paper addresses matching problems on vertex-weighted graphs. The vertex weights induce edge weights; we are primarily interested in the case where the weight of an edge between a vertex u and a vertex v is taken to be the sum of the weights of u and v. A matching M of a vertex-weighted graph is an *MWM* (resp., *MWMCM*) of G if it is an MWM (resp., MWMCM) of the corresponding edge-weighted graph.

A graph G is bipartite if the vertex set of G can be partitioned into two sets U and V such that every edge of G has one endpoint in U and one endpoint in V. In the present paper, we address bipartite graph problems where a particular bipartition of the vertices is specified as part of the input. Throughout the remainder of the paper, we use the term *bipartite graph* to refer to a triple (U, V, E) where U is a set of "left" vertices, V is a set of "right" vertices, and every edge in E has one endpoint in U and one endpoint in V.

The primary goal of this paper is to develop fast MWM algorithms for certain classes of vertex-weighted bipartite graphs. We analyze our algorithms in the RAM model, and we assume that each vertex weight can be represented using a constant number of machine words. It will prove to be useful to first develop fast algorithms for simpler problems in which the weights of either the left vertices, or the right vertices, are effectively zeroed out. With this in mind, we define an *LMWM* (resp., *RMWM*) of a vertex-weighted bipartite graph G as an MWM of the corresponding edge-weighted graph where the weight of an edge between a left vertex u and a right vertex v is given by the weight of u (resp., v). The terms *LMWMCM* and *RMWMCM* are defined analogously.

It is easy to see that we can compute an LMWM of a given bipartite graph $G = (U, V, E)$ by first deleting all of the negative-weight left vertices, and then computing an LMWM of the resulting bipartite graph.

Given a matching M of a bipartite graph G that is not an MCM of G, Berge's lemma [1, Theorem 1] implies the existence of a matching M' of G such that $|M'| = |M| + 1$ and the set of vertices matched in M is properly contained in the set of vertices matched in M'. Applying this idea repeatedly, we find that if M is a matching of a bipartite graph $G = (U, V, E)$, there is an MCM M' of G that matches all of the vertices matched in M. It follows that if every left vertex has nonnegative weight, then any LMWMCM is an LMWM.

Combining the observations of the two preceding paragraphs, we see that an LMWM of a given bipartite graph $G = (U, V, E)$ can be obtained by deleting all of the negative-weight left vertices, and then computing an LMWMCM of the resulting bipartite graph. Thus the LMWM and LMWMCM problems are essentially the same. In the remainder of the paper, we discuss only the LMWMCM problem. Symmetric remarks hold for the RMWM and RMWMCM problems.

Spencer and Mayr [18] attribute the following lemma, which is straightforward to prove, to Mendelsohn and Dulmage [12]; Spencer and Mayr also provide a proof.

Lemma 1. *Let M and M' be two MCMs of a bipartite graph $G = (U, V, E)$. Then there is an MCM of G that matches the set of left vertices matched in M to the set of right vertices matched in M'.*

Lemma 1 plays an important role in the Spencer-Mayr vertex-weighted matching framework discussed in Appendix A of [14], since it yields a reduction from the problem of vertex-weighted bipartite matching to the restricted case in which only the vertices on one side of the bipartition have nonzero weight. This reduction is restated below using the terminology of the present paper.

Lemma 2. *Let M be an LMWMCM of a vertex-weighted bipartite graph $G = (U, V, E)$, let U' be the set of left vertices of G that are matched in M, and let G' be the subgraph of G induced by $U' \cup V$. Then any RMWMCM of G' is an MWMCM of G.*

Proof. Immediate from Lemma 1. □

For any class C of graphs, and any integers m and n, we let $C_{m,n}$ denote the set of all graphs in C with at most m edges and at most n vertices.

A *representation scheme* ξ for a class C of graphs specifies a set *reps*(ξ, G) of possible representations for any given graph G in the class.

Let ξ denote a representation scheme for a class C of graphs. Scheme ξ is said to have *space complexity* at most $f(m, n)$ if, for any graph G in $C_{m,n}$, the space used by any representation in *reps*(ξ, G) is at most $f(m, n)$. Thus, for example, the standard adjacency list representation scheme has space complexity $O(m + n)$. Scheme ξ is said to have *MCM complexity* at most $f(m, n)$ if there is an $f(m, n)$-time algorithm which, given any representation in *reps*(ξ, G) of a graph G in $C_{m,n}$, computes an MCM of G. The *MWM* (resp., *LMWMCM*, *RMWMCM*, *MWMCM*) *complexity* of ξ is defined similarly, except that the input to the $f(m, n)$-time algorithm also specifies the vertex weights.

We say that a class C of graphs is *hereditary* if any induced subgraph of a graph in C also belongs to C. A representation scheme for a hereditary class C of graphs has *induced subgraph complexity* at most $f(m, n)$ if there is an $f(m, n)$-time algorithm which, given any representation in *reps*(ξ, G) of a graph $G = (V, E)$ in $C_{m,n}$, and any specified subset V' of V, computes a representation in *reps*(ξ, G') where G' denotes the subgraph of G induced by V'.

Lemma 3. *Let ξ denote a representation scheme for a hereditary class of bipartite graphs. If ξ has induced subgraph, LMWMCM, and RMWMCM complexity at most $f(m, n)$, then ξ has MWMCM complexity $O(f(m, n))$.*

Proof. Immediate from Lemma 2. □

Let ξ be a representation scheme for a class C of bipartite graphs. Scheme ξ is said to have *left-to-right search complexity* at most $f(m, n)$ if there exists an $f(m, n)$-time algorithm which, given any representation in *reps*(ξ, G) of a graph G in $C_{m,n}$, and any matching M of G, computes the set of all vertices that are reachable from some unmatched left vertex via an alternating path of unmatched and matched edges. The *right-to-left search complexity* of ξ is defined symmetrically. The *search complexity* of ξ is at most $f(m, n)$ if the left-to-right search complexity and right-to-left search complexity of ξ are each at most $f(m, n)$.

3 Ordered Representation Schemes

An *ordering* of a bipartite graph G specifies a total order over the set of left vertices of G, and a total order over the set of right vertices of G.

A representation of a bipartite graph G is *ordered* if it specifies an ordering of G, and allows the relative order of any two left (resp., right) vertices to be determined in constant time. A representation scheme ξ for a class C of bipartite graphs is *ordered* if for every G in C, every representation in $\mathit{reps}(\xi, G)$ is ordered.

Let ξ be an ordered representation scheme for a hereditary class C of bipartite graphs. We say that ξ has *left-to-right delete-min complexity* $f(n)$ if there exists an $f(n)$-time algorithm A which, given a representation R in $\mathit{reps}(\xi, G)$ for some graph G in C with at most n vertices, and a left vertex u of G, performs the following operation, which we denote *delete-min*(u): (1) if u has one or more incident edges, then letting v denote the least right vertex adjacent to u (with respect to the total order defined over the right vertices), and letting G' denote G with right vertex v removed, A returns v and modifies R to obtain a representation in $\mathit{reps}(\xi, G')$; (2) if u has no incident edges, then A returns *nil* and leaves R unchanged. The *right-to-left delete-min complexity* of ξ is defined symmetrically, along with the associated operation *delete-min*(v). The *delete-min complexity* of ξ is at most $f(n)$ if the left-to-right and right-to-left delete-min complexity of ξ are each at most $f(n)$. The following lemma is straightforward.

Lemma 4. *Let ξ be an ordered representation scheme for a hereditary class of bipartite graphs. If ξ has left-to-right (resp., right-to-left) delete-min complexity $f(n)$, then ξ has left-to-right (resp., right-to-left) search complexity $O(nf(n))$. Thus if ξ has delete-min complexity at most $f(n)$, then ξ has search complexity $O(nf(n)$.*

A bipartite graph is *chordal bipartite* if each cycle of length at least six has a chord. (Remark: A chordal bipartite graph need not be chordal because chordless cycles of length four are permitted.) The class of chordal bipartite graphs has been extensively studied, and various alternative characterizations are known. One such characterization is that a bipartite graph G is chordal bipartite if and only if G is $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$-free, which means that G admits an ordering such that for any pair of left vertices u and u' such that $u < u'$, and any pair of right vertices v and v' such that $v < v'$, if u is adjacent to v, u is adjacent to v', and u' is adjacent to v, then u' is adjacent to v'. In the present paper, we refer to such an ordering as an *elimination ordering.*

A representation of a chordal bipartite graph G is *elimination-ordered* if it specifies an elimination ordering of G. A representation scheme ξ for a class C of chordal bipartite graphs is *elimination-ordered* if for every graph G in C, each representation in $\mathit{reps}(\xi, G)$ is elimination-ordered.

Three lemmas related to elimination-ordered representation schemes are stated and proven in Appendix C of [14]. Lemma 11 of [14] is based on the MCM algorithm of Chang [3]. Lemmas 12 and 13 of [14] are useful for dealing with negative vertex weights.

4 Two-Directional Orthogonal Ray Graphs

A bipartite graph $G = (U, V, E)$ is called an *orthogonal ray graph (ORG)* if there exists a horizontal ray (i.e., a closed half-line parallel to the x-axis) corresponding to each left vertex, and a vertical ray (i.e., a closed half-line parallel to the y-axis) corresponding to each right vertex, such that a left vertex u is adjacent to a right vertex v if and only if the two corresponding rays intersect. If all of the horizontal rays go in the same direction (e.g., to the right), and all of the vertical rays go in the same direction (e.g., down), then we say that the ORG is a *two-directional ORG (2DORG)*

Various equivalent characterizations of the class of 2DORGs are known. Shrestha, Tayu, and Ueno [16] show that a graph is a 2DORG if and only if G admits a biadjacency matrix that is $\begin{bmatrix} * & 1 \\ 1 & 0 \end{bmatrix}$-free. In the notation of the present paper, this is equivalent to saying that G admits an ordering for which, for all pairs of left vertices u and u' such that $u < u'$, and all pairs of right vertices v and v' such that $v < v'$, if u is adjacent to v' and u' is adjacent to v, then u' is adjacent to v'. Notice that such an ordering is an elimination ordering, and hence every 2DORG is chordal bipartite. On the other hand, not every chordal bipartite graph is a 2DORG [17, Lemma 3.4.11].

Soto [17, Lemma 3.4.9] notes that a bipartite graph is a 2DORG if and only if it is a bicolored 2D-graph, that is, there exists a red point in the plane for each left vertex, and a blue point in the plane for each right vertex, such that there is an edge from left vertex u to right vertex v if and only if each component of the red point associated with u is at most the corresponding component of the blue point associated with v. This characterization suggests a natural *bicolored 2D representation* of a 2DORG in which each vertex is represented by a red or blue point in the x-y plane. (The original definition also suggests such a representation, where the point corresponding to a vertex is given by the endpoint of the corresponding ray.)

Soto [17, Lemma 3.4.1] also points out that every bicolored 2D-graph admits a *bicolored rook representation*, that is, a bicolored 2D representation satisfying the following additional constraints, where n denotes the number of vertices in the graph: (1) no two of the n points share a common x-value, or a common y-value; (2) the points are all drawn from the set $[n]^2$, where n denotes the number of vertices and $[n]$ denotes $\{i \mid 0 \leq i < n\}$. Furthermore, as shown by Soto, such a bicolored rook representation can be obtained from any bicolored 2D representation in $O(n \log n)$ time using a straightforward sorting procedure.

We now define a useful elimination-ordered representation scheme, denoted ξ^*, for the class of 2DORGs. Under scheme ξ^*, our representation of an n-vertex 2DORG G consists of a bicolored 2D representation plus two additional data structures. Like a bicolored rook representation, we require the bicolored 2D representation of G to satisfy the constraint that no two of the n points share a common x-value, or a common y-value. However, we do not require all of the points to be drawn from $[n]^2$; instead, we enforce the relaxed requirement that each coordinate is an integer that can be stored in a constant number of machine words. Before describing the two additional data structures associated with our representation, we define an ordering of G and prove that it is an elimination ordering. We define a total order $<$ over the set of left vertices as follows: $u < u'$ if and only if $y(u') < y(u)$. We define a total order $<$ over the set of right vertices as follows: $v < v'$ if and only if $x(v) < x(v')$. The following lemma establishes that this ordering is an elimination ordering.

Lemma 5. *Let $G = (U, V, E)$ be a 2DORG, and let R belong to reps(ξ^*, G). If $u < u'$, $v < v'$, (u, v') belongs to E, and (u', v) belongs to E, then (u', v') belongs to E.*

Proof. We need to prove that $x(u') \leq x(v')$ and $y(u') \leq y(v')$.

Since (u', v) belongs to E, we have $x(u') \leq x(v)$. Since $v < v'$, we have $x(v) < x(v')$. Hence $x(u') < x(v')$.

Since (u, v') belongs to E, we have $y(u) \leq y(v')$. Since $u < u'$, we have $y(u') < y(u)$. Hence $y(u') < y(v')$. □

We now describe the two additional data structures associated with our representation of G under scheme ξ^*. These data structures may be viewed as special cases of the priority search tree data structure of McCreight [11]. The first is a red-black tree that stores all of the left vertices in increasing order with respect to the total order $<$. This red-black tree is augmented (see [4, Chapter 14] for an introduction to augmented binary search trees) by maintaining, at each node α, an integer "*min*" field equal to the minimum, over all left vertices u stored in the subtree rooted at node α, of $x(u)$. It is straightforward to maintain the *min* field while supporting the standard dictionary operations in logarithmic time. This data structure allows us to support the *delete-min*(v) operation in logarithmic time, where v is an arbitrary right vertex.

The second data structure is a similar red-black tree that stores all of the right vertices in increasing order with respect to the total order $<$. This red-black tree is augmented by maintaining, at each node α, an integer "*max*" field equal to the maximum, over all right vertices v stored in the subtree rooted at node α, of $y(v)$. As in the case of the first data structure, it is straightforward to maintain the *max* field supporting the standard dictionary operations in logarithmic time. This second data structure allows us to support the *delete-min*(u) operation in logarithmic time, where u is an arbitrary left vertex.

Lemma 6. *The representation scheme ξ^* for the class of 2DORGs has space and induced subgraph complexity $O(n)$, delete-min complexity $O(\log n)$, search and MCM complexity $O(n \log n)$, and LMWMCM, RMWMCM, MWMCM, and MWM complexity $O(n \log^2 n)$.*

Proof. The $O(n)$ bound on space complexity is immediate from the definition of ξ^*. For the $O(n)$ bound on induced subgraph complexity, notice that we can form each of the two augmented red-black tree data structures associated with a specified induced subgraph as follows: (1) traverse the corresponding red-black tree for the original graph to extract the desired sorted sequence of vertices; (2)

arrange this sorted sequence of vertices into a perfectly balanced red-black tree structure (e.g., the same structure as is achieved in a binary heap); (3) fill in the values of the auxiliary fields in a bottom-up manner.

As indicated in our description of ξ^*, the two augmented red-black tree structures allow us to support arbitrary *delete-min*(u) and *delete-min*(v) operations in logarithmic time. Thus the delete-min complexity of ξ^* is $O(\log n)$.

Lemma 4 implies that the search complexity of ξ^* is $O(n \log n)$, and Lemma 11 of [14, Appendix C] implies that the MCM complexity of ξ^* is $O(n \log n)$.

Applying Lemma 8 of [14, Appendix A] with $f(m, n) = O(n \log n)$, we find that the LMWMCM and RMWMCM complexity of ξ^* is $O(n \log^2 n)$. Applying Lemma 3 with $f(m, n) = O(n \log^2 n)$, we find that the MWMCM complexity of ξ^* is $O(n \log^2 n)$.

It remains to bound the MWM complexity of ξ^*. Let C denote the class of all 2DORGs, and let C' denote the class of all graphs G' of the form $\mathit{extend}(G, U', V')$ where $G = (U, V, E)$ belongs to C, U' is a subset of U, and V' is a subset of V. By applying Lemma 13 of [14, Appendix C] with $f_0(m, n) = f_1(m, n) = O(n)$, $f_2(m, n) = f_3(m, n) = O(n \log n)$, and $f_4(n) = f_5(n) = O(\log n)$, we find that there is an elimination-ordered representation scheme with dummies ξ' for C' with space and induced subgraph complexity $O(n)$, search complexity $O(n \log n)$, and delete-min complexity $O(\log n)$. Thus, reasoning in the same manner as we did above for ξ^*, we find that ξ' has MCM complexity $O(n \log n)$, and LMWMCM, RMWMCM, and MWMCM complexity $O(n \log^2 n)$.

Lemma 13 of [14] also implies that if we are given a representation in $\mathit{reps}(\xi^*, G)$ of a graph $G = (U, V, E)$ in C, a subset U' of U, and a subset V' of V, then we can compute a representation in $\mathit{reps}(\xi', G')$ where $G' = \mathit{extend}(G, U', V')$ in $O(n)$ time. Since ξ' has MWMCM complexity $O(n \log^2 n)$, Lemma 9 of [14, Appendix A] implies that ξ^* has MWM complexity $O(n \log^2 n)$. □

Theorem 1. *Assume that we are given a bicolored 2D-graph representation of an n-vertex, vertex-weighted 2DORG G such that each x-value, y-value, or weight is an $O(1)$-word integer. Then an MWM of G can be computed in $O(n \log^2 n)$ time.*

Proof. Since any two $O(1)$-word integers can be compared in constant time, we can construct a representation in $\mathit{reps}(\xi^*, G)$ in $O(n \log n)$ time. Since representation scheme ξ^* has MWM complexity $O(n \log^2 n)$ by Lemma 6, the claim of the theorem follows. □

References

1. Berge, C.: Two theorems in graph theory. Proceedings of the National Academy of Sciences 43, 842–844 (1957)
2. Brodal, G.S., Georgiadis, L., Hansen, K.A., Katriel, I.: Dynamic matchings in convex bipartite graphs. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 406–417. Springer, Heidelberg (2007)

3. Chang, M.-S.: Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In: Nagamochi, H., Suri, S., Igarashi, Y., Miyano, S., Asano, T. (eds.) ISAAC 1996. LNCS, vol. 1178, pp. 146–155. Springer, Heidelberg (1996)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
5. Dekel, E., Sahni, S.: A parallel matching algorithm for convex bipartite graphs and applications to scheduling. Journal of Parallel and Distributed Computing 1, 185–205 (1984)
6. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. Journal of Computer and System Sciences 30, 209–221 (1985)
7. Gallo, G.: An $O(n \log n)$ algorithm for the convex bipartite matching problem. Operations Research Letters 3, 31–34 (1984)
8. Glover, F.: Maximum matching in convex bipartite graphs. Naval Research Logistic Quarterly 14, 313–316 (1967)
9. Katriel, I.: Matchings in node-weighted convex bipartite graphs. INFORMS Journal on Computing 20, 205–211 (2008)
10. Lipski Jr., W., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. Acta Informatica 15, 329–346 (1981)
11. McCreight, E.M.: Priority search trees. SIAM Journal on Computing 14, 257–276 (1985)
12. Mendelsohn, N.S., Dulmage, A.L.: Some generalizations of the problem of distinct representatives. Canadian Journal of Mathematics 10, 230–241 (1958)
13. Plaxton, C.G.: Fast scheduling of weighted unit jobs with release times and deadlines. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 222–233. Springer, Heidelberg (2008)
14. Plaxton, C.G.: Vertex-weighted matching in two-directional orthogonal ray graphs. Technical Report TR–13–16, Department of Computer Science, University of Texas at Austin (September 2013)
15. Scutellà, M.G., Scevola, G.: A modification of Lipski-Preparata's algorithm for the maximum matching problem on bipartite convex graphs. Ricerca Operativa 46, 63–77 (1988)
16. Shrestha, A.M.S., Tayu, S., Ueno, S.: On orthogonal ray graphs. Discrete Applied Mathematics 158, 1650–1659 (2010)
17. Soto, J.A.: Contributions on Secretary Problems, Independents Sets of Rectangles and Related Problems. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology (June 2011)
18. Spencer, T.H., Mayr, E.W.: Node weighted matching. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 454–464. Springer, Heidelberg (1984)
19. Steiner, G., Yeomans, J.S.: A linear time algorithm for determining maximum matchings in convex, bipartite graphs. Computers and Mathematics with Applications 31, 91–96 (1996)
20. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of the ACM 22, 215–225 (1975)
21. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Information Processing Letters 6, 80–82 (1977)

Bounded Representations of Interval and Proper Interval Graphs[⋆]

Martin Balko[1,⋆⋆], Pavel Klavík[2,⋆⋆], and Yota Otachi[3]

[1] Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
balko@kam.mff.cuni.cz

[2] Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
klavik@iuuk.mff.cuni.cz

[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. Klavík et al. [arXiv:1207.6960] recently introduced a generalization of recognition called the *bounded representation problem* which we study for the classes of interval and proper interval graphs. The input gives a graph G and in addition for each vertex v two intervals $\mathfrak{L}_v$ and $\mathfrak{R}_v$ called *bounds*. We ask whether there exists a bounded representation in which each interval I_v has its left endpoint in $\mathfrak{L}_v$ and its right endpoint in $\mathfrak{R}_v$. We show that the problem can be solved in linear time for interval graphs and in quadratic time for proper interval graphs.

Robert's Theorem states that the classes of proper interval graphs and unit interval graphs are equal. Surprisingly, the bounded representation problem is polynomially solvable for proper interval graphs and NP-complete for unit interval graphs [Klavík et al., arxiv:1207.6960]. So unless P = NP, the proper and unit interval representations behave very differently.

The bounded representation problem belongs to a wider class of restricted representation problems. These problems are generalizations of the well-understood recognition problem, and they ask whether there exists a representation of G satisfying some additional constraints. The bounded representation problems generalize many of these problems.

1 Introduction

In the recent data-filled world, visualization and graph drawing is becoming an increasingly more important topic. One is frequently asked to work with a huge object and to understand its structure. In some cases, it is useful to visualize the object in a way which reveals its structure. A prime example of this is the class of *interval graphs* which is one of the oldest and best-understood graph classes.

[⋆] The first two authors are supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023.

[⋆⋆] Supported by Charles University as GAUK 196213.

An interval graph G can contain many edges, so a standard drawing is not very understandable. But it has an *interval representation* $\mathcal{R}$ which is a collection of closed intervals $\{I_v : v \in V(G)\}$ representing the vertices of the graph such that $I_u \cap I_v \neq \emptyset$ if and only if $uv \in E(G)$. This representation nicely describes the structure of the edges. We denote the class of interval graphs by INT.

Interval graphs were first introduced by Hajós [10] in 1957. They caught quickly an attention of many researchers, for instance Benzer [1] used them in his experimental study of the DNA structure. The first polynomial-time recognition algorithms were given already in 1960's [9,8]. After a decade, Booth and Lueker [3] finally described a linear-time recognition algorithm based on a new tree-structure called PQ-trees, applicable also to other problems such as planarity. Nowadays, there are over several hundred papers dealing with many aspects of interval graphs.

An interval representation is called *proper* if $I_u \subseteq I_v$ implies $I_u = I_v$, i.e., no interval is a proper subset of another interval. And it is called *unit* if all intervals are of unit length. We consider two important subclasses of interval graphs: *proper interval graphs* ($\mathsf{PROPER\ INT}$) are graphs which admit proper interval representations, and similarly for *unit interval graphs* ($\mathsf{UNIT\ INT}$). The well-known theorem of Roberts [19] states that $\mathsf{PROPER\ INT} = \mathsf{UNIT\ INT}$.

1.1 The Bounded Representation Problem

Several recent papers study *restricted representation problems* in which we ask whether there exists, say, an interval representation of an input graph G satisfying some additional constraints; see for example [2,11,12,16,17,18]. In this paper, we study for the classes INT and $\mathsf{PROPER\ INT}$ one such problem called *bounded representation*, recently introduced by Klavík et al. [13]. This problem is related to many other restricted representation problems; see Section 1.2 for details.

For an arbitrary interval I, we denote its left endpoint by $\ell(I)$ and its right endpoint by $r(I)$. Let $\mathfrak{L}_v$ and $\mathfrak{R}_v$ be two intervals defined for each $v \in V(G)$. A representation $\mathcal{R}$ is called a *bounded representation* if $\ell(I_v) \in \mathfrak{L}_v$ and $r(I_v) \in \mathfrak{R}_v$ for each $v \in V(G)$. The bounded representation problem is the following decision problem:

Problem: The Bounded Representation Problem – BoundRep($\mathcal{C}$)
Input: A graph G and two intervals $\mathfrak{L}_v$ and $\mathfrak{R}_v$ for each $v \in V(G)$.
Output: Is there a bounded representation $\mathcal{R}$ of the class $\mathcal{C}$?

In the further text, we refer to the intervals $\mathfrak{L}_v$ as *left bounds* and to the intervals $\mathfrak{R}_v$ as *right bounds*, or just simply *bounds*. See Fig. 1a for an example of an BoundRep instance. It is easy to see that the bounded representation problem generalizes recognition; if all the bounds are set to $(-\infty, +\infty)$, they pose no restriction at all. We also allow trivial bounds consisting of single points.

1.2 Other Restricted Representation Problems

We review other restricted representation problems and discuss their relation to the bounded representation problem. The problems were considered for different

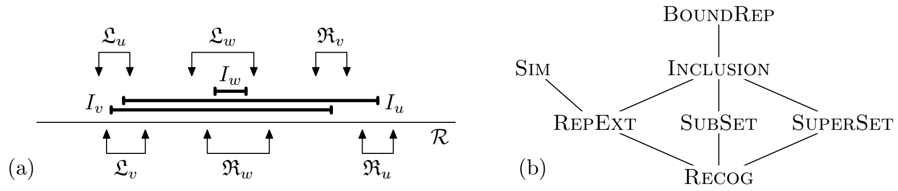


Fig. 1. (a) A bounded representation $\mathcal{R}$ of the class INT is given for a graph K_3. There exists no bounded proper interval representation since I_w is always a proper subset of I_u and I_v. (b) The Hasse diagram for different restricted representation problems. If $\mathcal{P} \le \mathcal{P}'$, then the problem $\mathcal{P}$ can be solved using the problem $\mathcal{P}'$.

intersection classes of graphs which we do not define formally; see the references for details. We note that all these problems generalize the recognition problem (Recog). See Fig. 1b for the relations between the problems.

Partial Representation Extension. This problem denoted by RepExt was introduced by Klavík et al. [16]. The input prescribes together with G an intersection representation $\mathcal{R}'$ of an induced subgraph G'. The goal is to find a representation $\mathcal{R}$ of the entire G which extends $\mathcal{R}'$, i.e., it assigns the same sets to the vertices of G' as $\mathcal{R}'$. The problem can be solved in polynomial time for interval graphs [2,15,16], proper and unit interval graphs [13], function and permutation graphs [12] and circle graphs [4]. For chordal graphs in the setting of subtree-in-a-tree graphs, several versions of the problem were considered in [14], and almost all of them are NP-complete. It is known that planar graphs have several intersection representations (contact representations of discs, etc.), but extending these representations is NP-hard [7].

The bounded representation problem generalizes partial representation extension, since one can prescribe singleton bounds for the intervals of G' and $(-\infty, +\infty)$ for the remaining bounds. (We note that the bounded representation problem can be considered also for many other classes of graphs.)

Inclusion Restrictions. Function graphs are intersection graphs of continuous functions defined on $[0,1]$. In [12], the following problem was considered and proved to be NP-complete. The input prescribes some functions partially, i.e., on partial domains $[a,b] \subseteq [0,1]$. The goal is to extend them to the full domain $[0,1]$.

We consider more generally three different problems Inclusion, SubSet, and SuperSet for interval graphs. In all problems, the input gives two intervals A_v and B_v for each vertex $v \in V(G)$. The goal is to construct a representation such that $A_v \subseteq I_v \subseteq B_v$. Further for SubSet, we put all $A_v = \emptyset$, and for SuperSet, we put all $B_v = (-\infty, \infty)$. It is easy to see that these problems can be reduced to the bounded representation problems, and Inclusion can solve RepExt.

Simultaneous Representations. This problem denoted by Sim was introduced and solved for several classes by Jampani et al. [11]. The input consists of two graphs G_1 and G_2 with some common vertices. The goal is to construct their representations $\mathcal{R}_1$ and $\mathcal{R}_2$ such that the common vertices are represented the same. Bläsius et al. [2] reduce RepExt(INT) to Sim(INT), and thus solve the first problem in linear time. On the other hand, when the problem is generalized to k input graphs, the best known result for many classes is an FPT algorithm in the number of common vertices based on the partial representation extension [16,4]. We are not aware of any relation of the simultaneous representations problem to the other considered problems.

Motivation. There are two very good motivations for studying the restricted representation problems. The first motivation is that they might be applicable. For instance, one might want to construct some specific representation of the given graph G. Using these restrictions, one can force the representation to be constructed in this way. The other motivation is that to solve these problems much better structural understanding is required. For classes like interval graphs, the structure of all representations is well understood and one can just use PQ-trees to solve the problems. For other classes like unit interval graphs [13] or circle graphs [4], the new structural results were developed which might be fruitful also for other purposes. In mathematics, it is generally desirable to have problems which force one to get better understanding of the objects.

1.3 Our Results

In this paper, we prove the following two theorems. Many details are omitted and for the full version see [arXiv:1309.1248]. For BoundRep(INT), we assume that the endpoints of the bounds are *sorted* from left to right, so we can work with the bounds efficiently. Otherwise, we need extra time $\mathcal{O}(n \log n)$ in the beginning.

Theorem 1. *The problem* BoundRep(INT) *with sorted endpoints of the bounds can be solved in time $\mathcal{O}(n+m)$ where n is the number of vertices and m is the number of edges.*

The algorithm of Theorem 1 is almost the same as the algorithm for RepExt(INT) of [15,16]. So the techniques developed for the partial representation extension problem can be directly applied to more general problems.

Theorem 2. *The problem* BoundRep(PROPER INT) *can be solved in time $\mathcal{O}(n^2)$ where n is the number of vertices.*

We note that it was already observed in [16] that the classes of proper and unit interval graphs behave differently with respect to the partial representation problem; unit interval graphs put additional restrictions is the form of precise rational positions. In [13], the problem RepExt(UNIT INT) was solved in quadratic time by linear programming. So it seemed that this difference is only in some additional numerical problems posed by unit intervals. Theorem 2 shows together with the result of [13, Proposition 2] that this understanding is fundamentally wrong (unless P = NP, of course):

Theorem 3 (Klavík et al. [13]). *The problem* BOUNDREP(UNIT INT) *is* NP-*complete.*

The problem is reduced from 3-partition and the hard part is to derive a correct ordering $\blacktriangleleft$ of the components from left to right; if the ordering is prescribed, one can solve the problem in quadratic time. The main difference for proper interval graphs is Proposition 2 which allows us to derive this ordering $\blacktriangleleft$. The remainder of the algorithm works similarly as in [13], only some places are more technical since we have to deal with both left and right bounds; in the case of unit interval graphs, we can work only with left bounds since the position $\ell(I_v)$ determines the position $r(I_v)$.

2 Preliminaries

For a graph G, we denote by $V(G)$ the set of its vertices and by $E(G)$ the set of its edges. We use $N[u]$ for the *closed neighborhood* of the vertex u, i.e, $N[u] = \{v \in V(G) : uv \in E(G)\} \cup \{u\}$.

A *(partial) ordering* is a transitive, reflexive and antisymmetric relation. A *pre-ordering* is just a transitive and reflexive relation, so several elements can be equal in a pre-ordering. An ordering/pre-ordering is called *linear* if every two elements are comparable.

For an arbitrary subset S of the real line, we define $\ell(S) = \inf\{x : x \in S\}$ and $r = \sup\{x : x \in S\}$. By $\lessdot$, we denote the *subset ordering* where $S \lessdot T$ for two subsets S and T if and only if $r(S) \le \ell(T)$; in other words S is completely on the left of T.

Endpoint Pre-orderings. There are two very natural ways how one can work with intervals and interval representations. The first option is to assign to each interval I two rational numbers $\ell(I)$ and $r(I)$. The second option, which we prefer in this paper, is just to consider the ordering $<$ of the endpoints as they appear from left to right. The reason is that this ordering contains all information about intersections of intervals; precise rational positions are not needed, we can just work with a topology of the representation. We note that this is not the case of unit interval representations, for which one has to consider precise rational number positions.

In the case of general interval graphs, one can assume that no two endpoints share their positions. For bounded representations, this is not true anymore since the bounds might force shared positions. In this case, $\le$ is a linear pre-ordering, with some sets of endpoints being equal in it. We say that an endpoint z is *in between* of x and y if $x \le z \le y$. If two endpoints x and y share position, we denote it by $x = y$, and by $x < y$ we denote that x is strictly on the left of y. It is important to state that if $x < y$, then one can add in between of x and y an arbitrary number of endpoints in any pre-ordering. If $x = y$, then only endpoints sharing the position with x and y can be added in between.

If we work with representations just as with left-to-right pre-orderings of the endpoints, then how can we decide whether the endpoints lie in the bounds?

Our assumption on the input is that we are given a linear pre-ordering of the endpoints of the bounds $\mathfrak{L}_v$ and $\mathfrak{R}_v$. The solution gives a bounded representation $\mathcal{R}$ in the form of a joined pre-ordering $\le$ of the endpoints of the bounds and the intervals. The bounds constraints just say that $\ell(\mathfrak{L}_v) \le \ell(I_v) \le r(\mathfrak{L}_v)$ and $\ell(\mathfrak{R}_v) \le r(I_v) \le r(\mathfrak{R}_v)$.

Simplifying Bounds. For each interval I_v, we want $\ell(I_v) \le r(I_v)$. So we assume each pair $\mathfrak{L}_v$ and $\mathfrak{R}_v$ satisfies $\ell(\mathfrak{L}_v) \le \ell(\mathfrak{R}_v)$ and $r(\mathfrak{L}_v) \le r(\mathfrak{R}_v)$. Otherwise we modify the instance by putting $\ell(\mathfrak{R}_v) := \ell(\mathfrak{L}_v)$, resp. $r(\mathfrak{L}_v) := r(\mathfrak{R}_v)$.

3 Bounded Representations of Interval Graphs

In this section we establish Theorem 1 which states that the problem BoundRep(INT) can be solved in time $\mathcal{O}(n + m)$ (given the pre-ordering of the endpoints of the bounds). First, we give a characterization of bounds for which the bounded representation exists. Then we describe the algorithm which checks this characterization, and since it is constructive, it can construct the bounded representation if it exists. We note that our approach is very similar to [15].

3.1 Characterization of Fulkerson and Gross

Fulkerson and Gross [8] gave the following characterization:

Lemma 1 (Fulkerson and Gross). *A graph G is an interval graph if and only if there exists a linear ordering $<$ of the maximal cliques of G such that for every vertex $v \in V(G)$ the cliques containing v appear consecutively in $<$.*

Proof (Sketch). We sketch this proof since it is important to understand the characterization. Let $\mathcal{R}$ be an interval representation. For each maximal clique C, we consider $\bigcap_{v \in C} I_v$, and according to Helly's theorem this intersection is non-empty. We pick an arbitrary point from this intersection, and we call it a *clique-point* and denote it by $\mathrm{cp}(C)$. Since these intersections are for different maximal cliques pairwise distinct, the clique-points are linearly ordered from left to right. It is routine to check that this is the ordering $<$ from Lemma 1.

On the other hand, given an ordering $<$ of the maximal cliques, we place clique-points arbitrarily in this ordering. Then for each vertex v, we put

$$\ell(I_v) = \min\{\mathrm{cp}(C) : v \in C\}, \qquad \text{and} \qquad r(I_v) = \max\{\mathrm{cp}(C) : v \in C\}, \quad (1)$$

i.e., we place I_v on top of the clique-points of cliques containing v. We obtain a correct interval representation of G. □

3.2 Orderings of Maximal Cliques Compatible with the Bounds

We want to construct a bounded representation in a similar manner, first by placing the clique-points from left to right and then by constructing the intervals using (1). But to ensure that the resulting representation is bounded, we cannot place the clique-points arbitrarily. For a maximal clique C, we denote by J_C the set of possible positions where $\text{cp}(C)$ can be placed; see the full version [arXiv:1309.1248] of this paper for the precise definition.

But now if $J_C \lessdot J'_C$, we know that $\text{cp}(C)$ has to be always placed on the left of $\text{cp}(C')$; so $\lessdot$ on the sets J_C gives the partial ordering of the cliques from left to right which we denote $\lessdot$ as well.

Proposition 1. *There exists a bounded representation $\mathcal{R}$ if and only if there is an ordering $<$ of the maximal cliques which is consecutive in every vertex v and extends $\lessdot$.*

Proof (Sketch). The constraints are necessary. We place the clique-points greedily from left to right according to the ordering $<$. When we place $\text{cp}(C)$, we place it on the right of the previously placed clique-point and in J_C. For contradiction suppose that no such point of J_C. We obtain a contradiction with the consecutivity property or the ordering $\lessdot$. □

3.3 The Algorithm

To solve BoundRep(INT), we proceed in the following main steps.

(1) We find maximal cliques of G, using the algorithm of Rose et al. [20] in time $\mathcal{O}(n+m)$.
(2) We compute the sets J_C. This can be done by a single sweep from left to right in time $\mathcal{O}(n+m)$. This gives us the partial ordering $\lessdot$ of maximal cliques according to which the clique-points have to appear on the real line.
(3) Test whether there is a linear ordering $<$ of the maximal cliques which extends $\lessdot$ and for each vertex the maximal cliques containing it appear consecutively. This can be done using [15, Section 2].
(4) If there is a suitable reordering $<$ of $\lessdot$, then we place the clique-points as in the proof of Proposition 1. Using (1) we construct a correct bounded representation $\mathcal{R}$ of G.

Note that if we only want to decide BoundRep(INT) without constructing a representation, then the last step can be omitted. For the first step, the input graph has to be chordal and then the total size of all cliques is $\mathcal{O}(n+m)$.

The constructed representation with $\text{cp}(C) \in J_C$ is correct since we have $\ell(I_v) \in \mathfrak{L}_v$ and $r(I_v) \in \mathfrak{R}_v$ for each $v \in V(G)$. Moreover $\mathcal{R}$ is an interval representation of G, since every clique-point lies exactly in the intervals representing the vertices of the corresponding maximal clique. Thus we can summarize the results.

Proof (Theorem 1). The proof follows the steps described in the beginning of the section. The correctness of the algorithm is ensured by Proposition 1. We already observed that for a given pre-ordering $\leq$ of the endpoints of the bounds from left to right, the construction of the reordering $<$ of $\lessdot$ can be done in time $\mathcal{O}(m+n)$. Since the representation $\mathcal{R}$ can be also constructed in linear time with respect to the size of G, we see that the whole algorithm runs in time $\mathcal{O}(m+n)$. □

4 Bounded Representations of Proper Interval Graphs

In this section, we establish Theorem 2 which states that the bounded representation problem of proper interval graphs can be solved in time $\mathcal{O}(n^2)$. Proper interval representations give two important orderings: the ordering $\blacktriangleleft$ of the components, and the ordering $\lhd$ of the intervals of the components. We first describe them in details and then we show how they can be used in solving of the BoundRep(PROPER INT) problem.

4.1 Component Orderings $\blacktriangleleft$

Let $\mathcal{R}$ be any representation of G and let C be a connected component. Then $\bigcup_{v \in C} I_v$ is a closed interval of the real line. Since the intervals corresponding to the components are pairwise disjoint, the components are ordered as $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$. Notice that for different representations we may get different orderings $\blacktriangleleft$, and when no restriction is posed on the representation, we can use each of the $c!$ possible orderings.

Suppose that $uv \notin E(G)$. We ask what conditions the bounds have to satisfy to determine that $I_u \lessdot I_v$ in any bounded representation of G. Since the intervals I_u and I_v do not intersect, it is sufficient to prove that $\ell(I_u)$ is always to the left of $r(I_v)$. This is clearly satisfied if and only if $\mathfrak{L}_u \lessdot \mathfrak{R}_v$.

For a given instance of the bounded representation problem, our goal is to determine some ordering $\blacktriangleleft$ in which a bounded representation exists. To do so, we derive a relation $\blacktriangleleft'$ such that the ordering $\blacktriangleleft$ of every bounded representation $\mathcal{R}$ has to extend $\blacktriangleleft'$. Let C and C' be two distinct components of G. We put $C \blacktriangleleft' C'$ if there exists a pair $u \in C$ and $v \in C'$ such that $\mathfrak{L}_u \lessdot \mathfrak{R}_v$.

The following proposition states that respecting the ordering $\blacktriangleleft'$ is already sufficient for solving the bounded representation problem:

Proposition 2. *A bounded representation of G in an ordering $\blacktriangleleft$ extending $\blacktriangleleft'$ exists if and only if there exists a bounded representation of G.*

Proof (Sketch). We argue only the non-obvious direction. Suppose that C and C' are two components incomparable in $\blacktriangleleft'$. In such a case, their bounds have to be hugely overlapping. There are two cases one has to deal with:

– All bounds of C and C' are pairwise intersecting. Then due to Helly's theorem, we can represent C and C' in any ordering in this intersection.
– Only bounds of, say, C are pairwise intersecting. But then due to Helly's theorem, we can represent C either on the left of the left-most bound of C', or on the right of the right-most bound of C. We still leave enough space for C' to be represented.

Then we repeatedly apply this local reordering of incomparable components till we modify the given bounded representation in the prescribed ordering $\blacktriangleleft$ which extends $\blacktriangleleft'$. □

We note that a similar proposition is not correct for unit interval graphs. The problem is that a component has some minimal size which it requires in every representation, so it cannot be placed in this arbitrary small common intersection of the bounds. Actually Klavík et al. [13, Theorem 1] proved that finding the correct ordering $\blacktriangleleft$ is the NP-complete part of the problem BoundRep(UNIT INT). For a prescribed ordering $\blacktriangleleft$, one can solve the bounded representation problem of unit interval graphs in quadratic time.

4.2 Vertex Orderings $\trianglelefteq$

Two vertices u and v are called *indistinguishable* if $N[u] = N[v]$. So being indistinguishable defines an equivalence relation on $V(G)$, and the classes of this equivalence are called *groups of indistinguishable vertices*. For every intersection representation, the vertices of each group can be represented the same, and so indistinguishable vertices are not very interesting from the structural point of view. This is not the case for the bounded representation problem (or any other problem of restricted representation), in which indistinguishable vertices can be given distinct bounds and thus are forced to be represented differently.

Vertex Orderings. Let $\mathcal{R}$ be any proper interval representation, and assume for a second that no two intervals of $\mathcal{R}$ are the same. Then the intervals are ordered from left to right, and we denote this ordering by $\vartriangleleft$. The ordering $\vartriangleleft$ is the ordering of the left endpoints, and at the same time the ordering of the right endpoints. In $\vartriangleleft$, each group of indistinguishable vertices has to appear consecutively. Deng et al. [6] characterize possible orderings $\vartriangleleft$ for connected proper interval graphs:

Lemma 2 (Deng et al.). *For a connected proper interval graph, the ordering $\vartriangleleft$ is uniquely determined up to local reordering of the groups of indistinguishable vertices and the complete reversal.*

In other words, there exists a partial ordering $<$ in which exactly the pairs of indistinguishable vertices are incomparable. Then each $\vartriangleleft$ is a linear extension of $<$ or its reversal. Corneil et al. [5] describe a simple linear-time algorithm for computing $<$.

Now we allow having several same intervals in the representation $\mathcal{R}$ since the bounds might force this situation. The representation $\mathcal{R}$ then gives a linear pre-ordering $\trianglelefteq$. When we construct bounded representations, we place intervals as the same if and only if this is forced by the bounds. It is easy to observe that if $I_u = I_v$, then the vertices u and v are indistinguishable.

Constraints Given by Bounds. In the case of bounded representations, the order of some pairs of the indistinguishable vertices can be prescribed by the bounds. Suppose that we restrict ourself to just a single component C of the input graph G and ignore the rest. Similarly to above, we produce a relation $\vartriangleleft'$ of the vertices of C.

Let u and v be two indistinguishable vertices of C. We put $u \trianglelefteq' v$ if and only if $\mathfrak{L}_u \lessdot \mathfrak{L}_v$ or $\mathfrak{R}_u \lessdot \mathfrak{R}_v$; so $\trianglelefteq'$ is a union of the subset order $\lessdot_\ell$ of the left bounds and the subset order $\lessdot_r$ of the right bounds. Notice that the pre-ordering $\trianglelefteq'$

does not have to be a partial ordering and that $u \trianglelefteq' v$ implies $u \trianglelefteq v$ for any representation $\mathcal{R}$.

Now, since we do not want to work with pre-ordering $\trianglelefteq$, we construct a reduced graph C' with modified bounds. The following proposition states that this construction does not change solution of the problem.

Proposition 3. *There exists a bounded representation of C with an ordering extending $<$ if and only if there exists a bounded representation of C' in an ordering $\lhd$ which extends both $<$ and $\lhd'$.*

Proof (Sketch). The construction of C' is done in two steps. First, we consider strongly connected components defined by $\trianglelefteq'$, and they have to be represented by the same intervals. Therefore, we unify the bounds of their intervals to force this. To prove the correctness, we reorder groups in C according to $\lhd$. For each group, we apply a similar greedy procedure as in Proposition 1. □

4.3 The Algorithm

The algorithm works as follows:

(1) We compute the ordering $\blacktriangleleft'$ of components, and construct a linear ordering $\blacktriangleleft$ extending $\blacktriangleleft'$.
(2) We proceed the components according to $\blacktriangleleft$ from left to right: $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$.
(3) When processing the component C_i:
   - Compute the partial ordering $<$, using [5].
   - For $<$ and its reversal do the following: for each group Γ of indistinguishable vertices, compute $\lhd'$, its strongly connected components, the reduced graph C'_i and its ordering $\lhd$.
   - Place the endpoints according to $\lhd$ from left to right, on the right side of the representation of C_{i-1} greedily as far to the left as possible.
   - Construct a representation of C_i, by copying the intervals I_{S_i}.

It remains to argue details concerning specific implementation and correctness which is easily implied by Proposition 2 and Proposition 3. See the full version [arXiv:1309.1248] for details.

5 Conclusions

In this paper, we give a polynomial time algorithm for the classes of interval and proper interval graphs for a recently introduced problem BoundRep. The main result of this paper is a rather surprising discovery that the bounded representation problem distinguishes the classes of proper and unit interval graphs: BoundRep(PROPER INT) is polynomially solvable but BoundRep(UNIT INT) is NP-complete [13]. We believe that is a very interesting problem to further investigate differences between the structures of proper and unit interval representations; this paper gives a good reason to do so.

Open Problems. We conclude with two open problems.

Problem 1. Is it possible to solve BOUNDREP(PROPER INT) in time $\mathcal{O}(n+m)$ (with a given left-to-right ordering of the bounds)?

The current bottleneck of our algorithm is the computation of $\trianglelefteq$ from $\trianglelefteq'$ which is the only step requiring time $\mathcal{O}(n^2)$.

Problem 2. What is the complexity of the BOUNDREP problem for other classes such as circular-arc graphs, circle graphs?

Currently, the only known results are for the classes INT, PROPER INT, and UNIT INT. Even attacking some simpler problems for these classes might be very interesting. For instance, solving the partial representation extension problem for circular-arc graphs could be a major advancement in the area of the restricted representation problems.

References

1. Benzer, S.: On the topology of the genetic fine structure. Proc. Nat. Acad. Sci. U.S.A. 45, 1607–1620 (1959)
2. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. In: SODA 2013 (2013)
3. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. J. Comput. System Sci. 13, 335–379 (1976)
4. Chaplick, S., Fulek, R., Klavík, P.: Extending partial representations of circle graphs. In: Wolff, A. (ed.) GD 2013. LNCS, vol. 8242, pp. 131–142. Springer, Heidelberg (accepted 2013)
5. Corneil, D.G., Kim, H., Natarajan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. Inf. Process. Lett. 55(2), 99–104 (1995)
6. Deng, X., Hell, P., Huang, J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. SIAM J. Comput. 25(2), 390–403 (1996)
7. Dorbec, P., Kratochvíl, J., Montassier, M.: Contact representations of planar graph: Rebuilding is hard (submitted, 2013)
8. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. 15, 835–855 (1965)
9. Gilmore, P.C., Hoffman, A.J.: A characterization of comparability graphs and of interval graphs. Can. J. Math. 16, 539–548 (1964)
10. Hajós, G.: Über eine Art von Graphen. Internationale Mathematische Nachrichten 11, 65 (1957)
11. Jampani, K., Lubiw, A.: Simultaneous interval graphs. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 206–217. Springer, Heidelberg (2010)
12. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 671–682. Springer, Heidelberg (2012)
13. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. In: Preparation (2012)

14. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T.: Extending partial representations of subclasses of chordal graphs. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 444–454. Springer, Heidelberg (2012)
15. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T., Vyskočil, T.: Linear-time algorithm for partial representation extension of interval graphs. CoRR abs/1306.2182 (2013)
16. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 276–285. Springer, Heidelberg (2011)
17. Köbler, J., Kuhnert, S., Watanabe, O.: Interval graph representation with given interval and intersection lengths. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 517–526. Springer, Heidelberg (2012)
18. Pe'er, I., Shamir, R.: Realizing interval graphs with size and distance constraints. SIAM J. Discrete Math. 175, 349–372 (1997)
19. Roberts, F.S.: Indifference graphs. Proof Techniques in Graph Theory, 139–146 (1969)
20. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SICOMP 5(2), 266–283 (1976)

Detecting and Counting Small Pattern Graphs

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

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

Abstract. We study the induced subgraph isomorphism problem and the general subgraph isomorphism problem for small pattern graphs.

We present a new general method for detecting induced subgraphs of a host graph isomorphic to a fixed pattern graph by reduction to polynomial testing for non-identity with zero over a field of finite characteristic. It yields new upper time bounds for several pattern graphs on five vertices and provides an alternative combinatorial method for the majority of pattern graphs on four and three vertices. Since our method avoids the large overhead of fast matrix multiplication, it can be of practical interest even for larger pattern graphs.

Next, we derive new upper time bounds on counting the number of isomorphisms between a fixed pattern graph with an independent set of size s and a subgraph of the host graph. We also consider a weighted version of the counting problem, when one counts the number of isomorphisms between the pattern graph and lightest subgraphs, providing a slightly slower combinatorial algorithm.

1 Introduction

The problems of detecting subgraphs or induced subgraphs of a graph that are isomorphic to another given graph are classical in algorithmics. They are generally termed as *subgraph isomorphism* and *induced subgraph isomorphism* problems, respectively. In particular, they include special cases such as well-known NP-hard problems as the independent set, clique, Hamiltonian cycle or Hamiltonian path problems.

Recently, the detection and/or counting variants of subgraph isomorphism and/or induced subgraph isomorphism have found several applications, e.g., in bio-molecular networks [1], social networks [18], automatic design of processor systems and network security [10]. In these applications pattern graphs are typically of fixed size which allows for polynomial-time solutions.

The fastest known general algorithms for the detection and counting variants of subgraph isomorphism and induced subgraph isomorphism, where the pattern graph has k vertices while the host graph has n vertices, run in time $O(n^{\omega(\lfloor k/3\rfloor,\lceil (k-1)/3\rceil,\lceil k/3\rceil})$

⋆ Research supported by the grant no. \ N206 566740 of the National Science Center.
⋆⋆ Research supported in part by Swedish Research Council grant 621-2011-6179.

[7,12,16], where $\omega(p,q,r)$ denotes the exponent of fast matrix multiplication for rectangular matrices of size $n^p \times n^q$ and $n^q \times n^r$, respectively [15]. For special graph classes faster algorithms are known (e.g., see [9,13,22]).

In the first part of our paper (Section 3), we study the detection variant of the induced subgraph isomorphism problem for pattern graphs of fixed size k, while in the second part (Section 4), we study counting variants of the general subgraph isomorphism problem for such pattern graphs. We denote the number of vertices and edges in the host graph by n and m.

Detection of Small Induced Subgraphs: In the literature, besides the naive $O(n^k)$-time method for the induced subgraph isomorphism, the method reducing the problem to triangle detection, or counting, respectively, is known when the pattern graph is an *arbitrary* fixed graph on k vertices. The underlying triangle problem can be solved by fast (rectangular) matrix multiplication which yields the upper bound of $O(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$ for the induced subgraph isomorphism [7,12,16]. Because fast matrix multiplication algorithms rely on algebra, the aforementioned method can be classified as non-combinatorial. Since fast matrix multiplication algorithms involve large overheads, the method is not very practical. The other drawback is that it is not sensitive to the topology of the pattern graph and yields the same upper bound for any k-vertex pattern graph (e.g., K_k, i.e., the k-clique).

There are a few known examples of pattern graphs of fixed size k for which one succeeded to design specific algorithms for induced subgraph isomorphism yielding asymptotic time upper bounds in terms of n lower than those offered by the aforementioned triangle based method. The oldest and most striking example is P_4, a path on four vertices, which can be detected in $O(n+m)$ time [3]. The other is P_3, the path on three vertices which can be also detected in $O(n+m)$ time [21], the third example is the diamond, obtained by removing a single edge from K_4, which can be detected in $O(n+m^{3/2})$ time [7] (cf. [12]). The fourth example is a paw which is a triangle connected to the fourth vertex by an edge, i.e., K_3+e. It can be detected in $O(n^{2.373})$ time [17,20]. (Analogous upper bounds hold for the pattern graphs that are the complement to one of the aforementioned pattern graphs.) Furthermore, an induced subgraph isomorphic to the generalized diamond $K_k - e$, i.e., K_k with a single edge removed, as well as an induced subgraph isomorphic to the path on k vertices, P_k, can be detected in $O(n^{k-1})$ time [11,21] which improves the triangle based bound from [7] for $k \leq 5$. For recent relative hardness results on detection of specific pattern graphs (e.g., $P_k,\ K_{p,q},\ K_{k-1}+e,\ K_k - e,\ C_4$) in the induced setting, see [8].

The triangle based method was refined by the use of fast rectangular matrix multiplication in [7] a decade ago. Since then no new general approaches to induced subgraph isomorphism for fixed pattern graphs have been presented.

We present a new framework for detecting an induced subgraph of fixed size k in a host graph on n vertices (Section 3). We associate a multivariate polynomial to a family of pattern graphs on k vertices that share both a subgraph on l vertices and the edges between the common subgraph and the remaining $k-l$ vertices outside the subgraph. The monomials of the polynomial are in one-to-one correspondence with the pattern graphs in the family and their coefficients are computed on the basis of the corresponding pattern graphs. If all the coefficients but one share a prime factor p, we can detect the pattern graph corresponding to the monomial whose coefficient is *not* divisible by p, by verifying the polynomial for non-identity with zero over a field of characteristic p. The crucial part of our proof is showing that the polynomial can be evaluated in $O(n^{l+1})$ time, which enables us to use the DeMillo-Lipton-Schwartz-Zippel lemma for the verification of the polynomial.

By applying our method, we can list eighteen pattern graphs on five vertices that can be detected in $O(n^4)$ time. With the exception of P_5 [11] and $K_5 - e$ [21], our upper bounds of $O(n^4)$ are new and in particular improve the bounds yielded by the triangle based method (Corollary 1). We can obtain also the upper time bound of $O(n^{k-1})$ for plenty of pattern graphs on $k > 5$ vertices. Although for so large pattern graphs, we cannot improve the upper bounds yielded by the triangle based method, the application of our combinatorial method not relying on fast matrix multiplication can be still be of practical interest.

For all graphs on four vertices except K_4, $K_{1,3}$ and C_4, and their complements, our method yields the upper bound of $O(n^3)$, which is better than that yielded by the triangle based method. Although our upper bounds for pattern graphs on four vertices do not improve the known bounds based on different specific methods [7,12,17], they have the advantage of not relying on the fast matrix multiplication algorithms and of being sensitive to the topology of the pattern graph. Similarly, for pattern graphs on three vertices, our method yields the upper bound of $O(n^2)$ in all the cases for which an upper time bound lower than that yielded by the triangle based method is known.

Our main technical contribution in the first part of our paper is as follows.

Let $\mathcal{H}_k$ denote the family of single representatives of all isomorphism classes of undirected graphs on k vertices, and let $\mathcal{H}_k(l)$ stand for its subfamily comprised of all graphs in $\mathcal{H}_k$ having an independent set of size at least $k - l$. Consider $H \in \mathcal{H}_k(l)$ and an induced subgraph H_{sub} of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set. Let $\mathcal{H}_k(H_{sub}, H)$ stand for the family of all supergraphs H' of H (including H) such that H' has the same vertex set as H, H_{sub} is also an induced subgraph of H', and the set of edges with endpoints in both H_{sub} and $H' \setminus H_{sub}$ is the same as that with endpoints in both H_{sub} and $H \setminus H_{sub}$, see Fig 1.

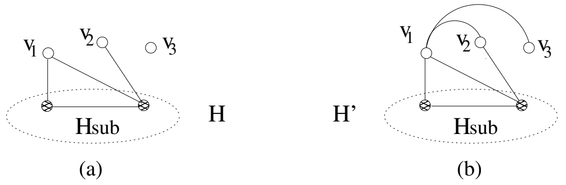


Fig. 1. (a) An example of a graph H composed of the induced subgraph H_{sub} and the vertex set $\{v_1, v_2, v_3\}$ that forms an independent set in H. (b) An example of a supergraph H' of H_{sub} in $\mathcal{H}_k(H_{sub}, H)$.

Finally, for each $H' \in \mathcal{H}_k(H_{sub}, H)$, let $B(H_{sub}, H, H')$ denote the number of isomorphisms between H_{sub} and an induced subgraph of H', say H^f_{sub}, that can be extended to an isomorphism between H and the subgraph of H' consisting of H^f_{sub}, all edges of H' incident to H^f_{sub} and all the remaining vertices of H'. We obtain the following result (Theorems 3, 4).

Let $F \in \mathcal{H}_k(H_{sub}, H)$, where $k = O(1)$. Suppose that there is a prime number p that is a factor of $B(H_{sub}, H, H')$ for all $H' \in \mathcal{H}_k(H_{sub}, H)$, except for $H' = F$. There is a randomized algorithm that detects if a graph on n vertices contains an induced subgraph isomorphic to F, with one-sided error probability polynomially small in n (i.e., $O(n^{-\alpha})$ for $\alpha > 1$), in $O(n^{l+1})$ time. Importantly, for $k - l = 2$, $\mathcal{H}_k(H_{sub}, H)$ contains two graphs and it is sufficient to require that p is a prime factor of the number of automorphisms for the other graph in $\mathcal{H}_k(H_{sub}, H)$ but it is not a prime factor of the number of automorphisms of F.

The idea of associating a polynomial over a finite field to the sought structure has been already used by Edmonds to detect a perfect matching [6]. It appears in several recent papers that exploit also the idea of monomial cancellation [2,14].

Counting Subgraph Isomorphisms for Fixed Pattern Graphs: Vassilevska and Williams studied the counting variant of subgraph isomorphism under the assumption that the k-vertex pattern graph has an independent set of size s [22]. They designed combinatorial algorithms (i.e., not relying on fast matrix multiplication) for this counting problem running in time $O(f(s)n^{k-s+2})$ where f is an exponential or super-exponential function depending only on s. Subsequently, Kowaluk et al. [13] designed an algorithm for the corresponding detection problem using fast rectangular matrix multiplication and running in time $O(n^{\omega(\lceil (k-s)/2 \rceil, 1, \lfloor (k-s)/2 \rfloor)}) \le O(n^{k-s+1})$ when $k = O(1)$. They also established an analogous upper bound for the counting variant when the size s of the independent set is 2.

By a *subgraph isomorphism* between the pattern graph and the host graph, we shall mean a one-to-one mapping of vertices in the pattern graph into vertices of the host graph that preserves vertex adjacency.

In the second part of our paper (Section 4), we present an algorithm for counting subgraph isomorphisms between a pattern graph with k vertices and an independent set of cardinality s and a host graph with n vertices. It runs in time $O(n^{\omega(\lceil (k-s)/2 \rceil, 1, \lfloor (k-s)/2 \rfloor)})$ which matches the upper bound for detection from [13] and largely extends that for counting showed only for $s \le 2$ in [13]. Our algorithm relies on a solution to the so called $(k-s)$-neighborhood problem from [13], which in turn relies on fast rectangular matrix multiplication.

We also consider a weighted version of the counting problem, where real weights are assigned to the edges and/or vertices of the host graph, and the task is to count the number of subgraph isomorphisms between the pattern graph with k vertices containing an independent set of cardinality s and the host graph with n vertices that minimize the total weight of the images of the pattern graph. For this more general counting problem we design a slightly slower combinatorial algorithm running in $O(n^{k-s+1} \log n)$ time when $k = O(1)$.

In the literature, various weighted versions of the counting variants of subgraph isomorphisms have been studied solely in terms of the sizes of the pattern and host graphs, and without any explicit assumption on the size of independent set in the pattern graph [4,22].

2 Preliminaries

An *isomorphism* between two graphs F and G is a one-to-one mapping f of the vertices of F onto vertices of G such that $\{u, v\}$ is an edge of F iff $\{f(u), f(v)\}$ is an edge of G. If $F = G$ then an isomorphism between F and G is called an *automorphism* of F. F is *isomorphic* to G if there is an isomorphism between F and G.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph G' of G is *induced* if $E' = (V' \times V') \cap E$.

A *subgraph isomorphism* between two graphs F and G is an isomorphism between F and a subgraph of G.

The *detection version* (or equivalently, the *decision version*) of the *subgraph isomorphism* problem is to decide for a host graph and a pattern graph if the host graph has a subgraph isomorphic to the pattern graph. The *counting version* of subgraph isomorphism asks for reporting the total number of subgraphs of the host graph isomorphic to the pattern graph or just the total number of subgraph isomorphisms between these two graphs. The corresponding versions of *induced subgraph isomorphism* are defined analogously by replacing "subgraph" with "induced subgraph".

Let S be a subgraph of G with an order on its l vertices, or just an ordered subset of l vertices of G. The *S-neighborhood type* of a vertex of G is a binary vector b with l coordinates such that $b(i) = 1$ iff v is adjacent to the i-th vertex of S for $i = 1, ..., l$.

The *l-neighborhood problem* is to determine, for each ordered l-tuple α of vertices of G and each binary vector b with l coordinates, the number of vertices v in G outside α whose α-neighborhood type is b.

Fact 1 [13]. *The l-neighborhood problem for a graph on n vertices can be solved in $O(n)$ time for $l = 1$ and in $O(2^l n^{\omega(\lceil l/2 \rceil, 1, \lfloor l/2 \rfloor)})$ time for $l \geq 2$.*

3 A New Method of Detecting Small Induced Subgraphs

Let H be a pattern graph on k vertices in $\mathcal{H}_k(l)$ (see the introduction) and let H_{sub} be an induced subgraph of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set. Recall the definition of the family $\mathcal{H}_k(H_{sub}, H)$ of supergraphs of H and the definition of the quantities $B(H_{sub}, H, H')$, for $H' \in \mathcal{H}_k(H_{sub}, H)$, given in the introduction. Let $\mathcal{SH}_k(H_{sub}, H)$ stand for the family of single representatives of all isomorphism classes in $\mathcal{H}_k(H_{sub}, H)$, i.e., one graph from each isomorphism class.

Finally, for a pattern graph H and a host graph G, let $SI(H, G)$ be the set of all subsets of $V(G)$ on $|H|$ vertices that induce a subgraph of G isomorphic to H. Next, let $PI(H, G)$ denote the multivariate polynomial $\sum_{S \in SI(H,G)} \prod_{v \in S} x_v$, and let

$$P(H_{sub}, H, G) = \sum_{H' \in \mathcal{SH}_k(H_{sub}, H)} B(H_{sub}, H, H') PI(H', G).$$

To state our key lemma, for a prime number p, $H \in \mathcal{H}_k(l)$ and $H_{sub} \in \mathcal{H}_l$, we define the subset $\mathcal{H}_k^p(H_{sub}, H)$ of $\mathcal{H}_k(H_{sub}, H)$ as the set of all $H' \in \mathcal{H}_k(H_{sub}, H)$ for which $B(H_{sub}, H, H')$ is divisible by p. Also, recall that the *characteristic* of a ring or a field is the minimum number of 1 in a sum of ones that yields 0.

Lemma 1. *Let $H \in H_k(l)$ and let H_{sub} be an induced subgraph of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set. For a prime number p, a host graph G contains an induced subgraph isomorphic to a graph in $\mathcal{H}_k(H_{sub}, H) \setminus \mathcal{H}_k^p(H_{sub}, H)$ iff the polynomial $P(H_{sub}, H, G)$ is not identical to zero over a field of characteristic p.*

Proof. All the monomials with coefficients $B(H_{sub}, H, H')$, where $H' \in \mathcal{H}_k^p(H_{sub}, H)$, vanish over any field of characteristic p. On the other hand, those with the coefficient $B(H_{sub}, H, H')$, where $H' \in \mathcal{H}_k(H_{sub}, H) \setminus \mathcal{H}_k^p(H_{sub}, H)$, (if any) remain with a non-zero coefficient equal to $B(H_{sub}, H, H') \mod p$. It follows from the definition of the polynomial $P(H_{sub}, H, G)$ that it is not identical to zero iff G contains an induced subgraph isomorphic to a graph in $\mathcal{H}_k(H_{sub}, H) \setminus \mathcal{H}_k^p(H_{sub}, H)$. □

The following lemma on polynomial identity testing has been shown independently by DeMillo and Lipton, Schwartz, and Zippel.

Lemma 2. *[5,19] Let $Q(x_1, x_2, ..., x_m)$ be a nonzero polynomial of degree d over a field of size r. Then, for f_1, f_2, ..., f_m chosen independently and uniformly at random from the field, the probability that $Q(f_1, f_2, ..., f_m)$ is not equal to zero is at least $1 - \frac{d}{r}$.*

The second part of Section 3 is devoted to the proof of the following key theorem.

Theorem 1. *Let p be a fixed prime number. For $H \in \mathcal{H}_k(l)$ and an induced subgraph H_{sub} of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set, the polynomial $P(H_{sub}, H, G)$ can be evaluated for a given assignment of values over a field $F_{p^{O(\log n)}}$ of characteristic p in $O(n^{l+1})$ time.*

By combining Lemmata 1, 2, and Theorem 1, we obtain our first main result.

Theorem 2. *Let $H \in \mathcal{H}_k(l)$, let H_{sub} be an induced subgraph of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set, and let p be a fixed prime number. There is a randomized algorithm that detects if a graph on n vertices contains an induced subgraph isomorphic to a graph in $\mathcal{H}_k(H_{sub}, H) \setminus \mathcal{H}_k^p(H_{sub}, H)$, with one-sided error probability polynomially small in n, in $O(n^{l+1})$ time.*

Proof. By Lemma 1, it is sufficient to show how to test if the polynomial $P(H_{sub}, H, G)$ is not identical to zero, with one-sided error probability polynomially small in n, in $O(n^{l+1})$ time.

Note that the polynomial $P(H_{sub}, H, G)$ is of degree k. We can use Lemma 2 with a field $F_{p^{c \log n}}$ of characteristic p to obtain a randomized test of the polynomial $P(H_{sub}, H, G)$ for not being identical to zero with one side error probability not larger than $\frac{k}{p^{c \log n}}$. For sufficiently large constant c, the error probability is not larger than $\frac{1}{n^\alpha}$, $\alpha > 1$.

By Theorem 1, the test can be performed in $O(n^{l+1})$ time. □

Theorem 3. *Let $H \in \mathcal{H}_k(l)$, let H_{sub} be an induced subgraph of H on l vertices such that the $k - l$ vertices in $H \setminus H_{sub}$ form an independent set, and let $F \in \mathcal{H}_k(H_{sub}, H)$,* *where $k = O(1)$. Suppose that there is a prime number p that is a factor of $B(H_{sub}, H, H')$ for all $H' \in \mathcal{H}_k(H_{sub}, H)$, except for $H' = F$. There is a randomized algorithm that detects if a graph on n vertices contains an induced subgraph isomorphic to F, with one-sided error probability polynomially small in n, in $O(n^{l+1})$ time.*

Note that in Theorem 3, if $k - l = 2$ then $\mathcal{H}_k(H_{sub}, H)$ contains two graphs.

Theorem 4. *Let $H \in H_k(k-2)$, let H_{sub} be an induced subgraph of H on $k-2$ vertices such that the two vertices in $H \setminus H_{sub}$ form an independent set, and let $F \in \mathcal{H}_k(H_{sub}, H)$, where $k = O(1)$. Suppose that there is a prime number p that is a factor of the number of automorphisms of the other $H' \in \mathcal{H}_k(H_{sub}, H)$ and that is not a prime factor of the number of automorphisms of F. There is a randomized algorithm that detects if a graph on n vertices contains an induced subgraph isomorphic to F, with one-sided error probability polynomially small in n, in $O(n^{k-1})$ time.*

Proof. Let $H' \in \mathcal{H}_k(H_{sub}, H)$, and let $\mathcal{F}$ be the set of all isomorphisms f between H_{sub} and an induced subgraph of H' satisfying the requirements from the definition of $B(H_{sub}, H, H')$.

Consider an extension f' of $f \in \mathcal{F}$ to an isomorphism between H and the subgraph of H' composed of H^f_{sub}, all edges of H' incident to H^f_{sub}, and all other vertices of H'. If $H' = H$ then f' is an automorphism of H'. Otherwise, H' is the other member of $\mathcal{H}_k(H_{sub}, H)$ obtained by adding the edge between the two independent vertices of H outside H_{sub}. Then, f' is also an automorphism of H' since the only edge in H' not incident to H^f_{sub} has to connect the images by f' of the aforementioned two independent vertices in H.

It follows that each $f \in \mathcal{F}$ can be identified with the class of all automorphisms of H' that are equal to each other on H_{sub}. Conversely, each such class yields a distinct member in $\mathcal{F}$.

We conclude that $B(H_{sub}, H, H')$ is equal to the number of automorphisms of H' divided by the number of automorphisms of H' that are identity on H_{sub}. It remains to observe that the latter number is the same for both members in $\mathcal{H}_k(H_{sub}, H)$. Simply, for each $H' \in \mathcal{H}_k(H_{sub}, H)$, the set of automorphisms of H' that are identity on H_{sub} contains either only the identity automorphism or also the automorphism that switches the two vertices of H' outside H_{sub} in case their H_{sub}-neighborhoods types are equal. □

Efficient Evaluation of $P(H_{sub}, H, G)$: We shall define a polynomial equivalent with $P(H_{sub}, H, G)$ (more precisely, a different decomposition of $P(H_{sub}, H, G)$) and show that it can be efficiently evaluated. To begin with, we need the following notation and lemma.

Let α be a fixed ordered l-tuple of vertices of the graph H that induces the subgraph H_{sub} and for any $b \in \{0,1\}^l$, let $\eta_{\alpha,H}(b)$ be the number of vertices of the α-neighborhood type b in H. By the definition of $\mathcal{H}_k(H_{sub}, H)$, we obtain the following lemma.

Lemma 3. *Let f be an isomorphism between a graph $H' \in \mathcal{H}_k(H_{sub}, H)$ and a subgraph of G induced by a set S of k vertices in $V(G)$ and let $\alpha_f = (f(\alpha_1), ..., f(\alpha_l))$. The number of vertices of the α_f-neighborhood type b in S equals $\eta_{\alpha,H}(b)$.*

For an ordered l-tuple γ of l different vertices in $V(G)$ and for $b \in \{0,1\}^l$, let $V(\gamma, b)$ be the set of all vertices of the γ-neighborhood type b in $V(G) \setminus \gamma$. The polynomial $Q(\gamma, H, G)$ is defined by

$$\prod_{i=1}^{l} x_{\gamma_i} \prod_{b \in \{0,1\}^l \& \eta_{\alpha,H}(b) \neq 0} \left(\sum_{U \subseteq V(\gamma,b) \wedge |U| = \eta_{\gamma,H}(b)} \prod_{v \in U} x_v \right).$$

Lemma 4. *Let γ be an ordered l-tuple of vertices in $V(G)$ inducing a subgraph of G isomorphic to H_{sub}, and let S be a set of k vertices in $V(G)$. The monomial $\prod_{v \in S} x_v$ occurs (exactly once) in $Q(\gamma, H, G)$ iff S includes the vertices of γ and there is an isomorphism between a graph in $\mathcal{H}_k(H_{sub}, H)$ and the subgraph of G induced by S that maps the i-th vertex of the l-tuple α on the i-th vertex of the l-tuple γ.*

Proof. To begin with, observe that each monomial of $Q(\gamma, H, G)$ is unique. If $\prod_{v \in S} x_v$ occurs in $Q(\gamma, H, G)$ then S has to include the vertices of γ by the definition of $Q(\gamma, H, G)$. Specify a mapping $g : V(H) \to S$ such that $g(\alpha_i) = \gamma_i$ for $i = 1, ..., l$, and for each $b \in \{0,1\}^l$, g maps the j-th vertex of the α-neighborhood type b in $V(H)$ onto the j-th vertex of the γ-neighborhood type b in S for $j = 1, .., \eta_{\alpha,H}(b)$ (for any orderings of the vertices of the respective type b). Now, observe that g defines an isomorphism between some graph in $\mathcal{H}_k(H_{sub}, H)$ and that induced by S in G.

Conversely, if there is an isomorphism f between some graph in $\mathcal{H}_k(H_{sub}, H)$ and the subgraph of G induced by S that maps α_i on γ_i for $i = 1, .., l$, then the l-tuple α_f in Lemma 3 equals γ and hence by this lemma and the definition of $Q(\gamma, H, G)$, $\prod_{v \in S} x_v$ occurs in $Q(\gamma, H, G)$. □

Let L be the set of all l-tuples γ of different l vertices such that there is an isomorphism between H_{sub} and the subgraph of G induced by γ that maps α_i on γ_i for $i = 1, ..., l$. Next, let $Q(H_{sub}, H, G) = \sum_{\gamma \in L} Q(\gamma, H, G)$.

The idea behind the following key lemma is as follows. Consider a monomial that corresponds to a subgraph G' of G isomorphic to $H' \in \mathcal{H}_k(H_{sub}, H)$ and occurs with the coefficient $B(H_{sub}, H, H')$ in $P(H_{sub}, H, G)$. It occurs once in each of the $Q(\beta, H, G)$ forming $Q(H_{sub}, H, G)$, where there is an extension of the function mapping the i-th vertex of α on the i-th vertex of β to a subgraph isomorphism between H and G', satisfying the requirements from the definition of $B(H_{sub}, H, H')$. Hence, the number of such β is $B(H_{sub}, H, H')$.

Lemma 5. *The equality $P(H_{sub}, H, G) = Q(H_{sub}, H, G)$ holds.*

Proof. By the definition of $P(H_{sub}, H, G)$, if $\prod_{v \in S} x_v$ is a monomial of this polynomial then there is an isomorphism f between a graph in $\mathcal{SH}_k(H_{sub}, H)$ and the subgraph of G induced by S. Let $\alpha_f = (f(\alpha_1), ..., f(\alpha_l))$. Then, $\prod_{v \in S} x_v$ is a monomial in $Q(\alpha_f, H, G)$ and hence also in $Q(H_{sub}, H, G)$.

Conversely, if $\prod_{v \in S} x_v$ is a monomial in $Q(H_{sub}, H, G)$, i.e., a monomial in $Q(\gamma, H, G)$ for some $\gamma \in L$, then it follows from Lemma 4 that the subgraph of G induced by S is isomorphic to a graph H' in $\mathcal{SH}_k(H_{sub}, H)$. Hence, it is also a monomial in $P(H_{sub}, H, G)$. To show the equality, it remains to show that the monomial occurs $B(H_{sub}, H, H')$ times in $Q(H_{sub}, H, G)$, i.e., that there are $B(H_{sub}, H, H')$ l-tuples β such that $\prod_{v \in S} x_v$ is a monomial in $Q(\beta, H, G)$.

An occurrence of $\prod_{v \in S} x_v$ as a monomial in $Q(\beta, H, G)$ is in one-to-one correspondence with the class of all subgraph isomorphisms between H and the subgraph G' of G induced by S that map α_i on β_i for $i = 1, ..., l$. It follows from the definition of $B(H_{sub}, H, H')$ that the number of such l-tuples β is equal to $B(H_{sub}, H, H')$. □

Proof of Theorem 1. By Lemma 5, it is sufficient to show that $Q(H_{sub}, H, G)$ can be evaluated over the field in the claimed time.

The set L of all ordered l-tuples γ of different l vertices such that there is an isomorphism between H_{sub} and the subgraph of G induced by γ that maps α_i on γ_i for $i = 1, ..., l$, can be easily computed in $O(l^2 l! n^l) = O(n^l)$ time.

For all $\gamma \in L$, and all $b \in \{0, 1\}^l$, we can compute the sets $V(\gamma, b)$ of vertices $v \in V$ that have γ-neighborhood of type b in $O(2^l l n^{l+1}) = O(n^{l+1})$ time in total.

By the definition of $Q(H_{sub}, H, G)$, it is sufficient to show that for an arbitrary $\gamma \in L$, the polynomial $Q(\gamma, H, G)$ can be evaluated in $O(n)$ time (recall that $k = O(1)$). This in turn by $l = O(1)$ reduces to showing that for an arbitrary $b \in \{0, 1\}^l$, the polynomial $\prod_{i=1}^{l} x_{\gamma_i} \sum_{U \subseteq V(\gamma, b) \wedge |U| = \eta_{\alpha,H}(b)} \prod_{v \in U} x_v$, can be evaluated in $O(n)$ time.

For $i = 1, ..., n$, let X_i be the set of variables $x_1, ..., x_i$. Next, for a positive integer q, let $C^q(X_i)$ denote the elementary symmetric polynomial of degree q, i.e., $\sum_{T \subseteq X_i \wedge |T| = q} \prod_{x_j \in T} x_j$. For convention, we let $C^0(X_i)$ to be 1 in the field. $C^q(X_n)$ can be evaluated for a given assignment of values over the field by the recurrence $C^q(X_{i+1}) = x_{i+1} C^{q-1}(X_i) + C^q(X_i)$.

For $q = O(1)$, we can evaluate all $C^q(X_i)$ using this recurrence by dynamic programming in lexicographic order of (q, i) in $O(n)$ time. It follows from $\eta_{\alpha,H}(b) = O(1)$ that the polynomial
$\prod_{i=1}^{l} x_{\gamma_i} \sum_{U \subseteq V(\gamma, b) \wedge |U| = \eta_{\alpha,H}(b)} \prod_{v \in U} x_v$ can be evaluated in $O(n)$ time. □

To formulate our corollary on applications of Theorem 4 to pattern graphs on five vertices, we need to introduce the following notation.

A graph on five vertices consisting of P_4 and a vertex adjacent to exactly one interior vertex of the P_4 is called a *chair*. Next, a graph consisting of C_4 and a vertex adjacent to exactly one vertex of the C_4 is called a *4-pan*.

Corollary 1. *The method of Theorem 4 can be used to detect $K_2 + 3K_1$, $2K_2 + K_1$, $P_3 + 2K_1$, $K_3 + e + K_1$, chair, $4 - pan$, $K_4 - e + K_1$, P_5, $P_3 + P_2$ and their respective complements, as induced subgraphs, with one-sided error probability polynomially small in n, in $O(n^4)$ time.*

With the exception of P_5 [11] and $K_5 - e$ [21], our upper bounds of $O(n^4)$ are new. For the applications of our method to pattern graphs on at most four vertices or on six vertices the reader is referred to the full version.

4 Counting Subgraph Isomorphisms

Our first method for counting subgraph isomorphisms relies on Fact 1, see Prel.

Theorem 5. *The number of subgraph isomorphisms between a fixed pattern graph with k vertices and with an independent set on s vertices and a host graph on n vertices can be computed in time $O(n^{\omega(\lceil(k-s)/2\rceil,1,\lfloor(k-s)/2\rfloor)})$.*

Proof. sketch. Let H be the pattern graph on k vertices with an independent set I on s vertices. Next, let H_{sub} be the subgraph of H induced by all its $l = k - s$ vertices outside I. For a vertex $v \in I$, let $b(v)$ denote the H_{sub}-neighborhood type of v. For $w \in \{0,1\}^l$, let t_w be the number of vertices v in I for which $b(v) = w$.

Let $G = (V, E)$ be the host graph on n vertices. Consider an ordered l-tuple α of vertices in G such that the function f_α mapping the i-th vertex of H_{sub} onto the i-th vertex of α is a subgraph isomorphism between H_{sub} and G.

We shall count the number of different subgraph isomorphisms between H and G that are extensions of the subgraph isomorphism f_α between H_{sub} and G.

For this purpose, we denote the α-neighborhood type of a vertex u in G by $b_\alpha(u)$. For $w \in \{0,1\}^l$, let $n_w(\alpha)$ be the number of vertices u in G outside α for which $b_\alpha(u) = w$. Also, for two vectors c, $d \in \{0,1\}^l$, we say that d *dominates* c if for $i = 1, ..., l$, $c_i \le d_i$. For a vector $c \in \{0,1\}^l$, the vector directly following c in the lexicographic order (if any) is denoted by $suc(c)$.

Observe that by our definitions, the number of extensions of the subgraph isomorphism f_α between H_{sub} and G to a subgraph isomorphism between H and G is equal to the number of ways we can choose for each $w \in \{0,1\}^l$ an ordered tuple of t_w vertices u in G such that $b_\alpha(u)$ dominates w. It is sufficient to consider solely those w for which $t_w \neq 0$, i.e., there are vertices in $H \setminus H_{sub}$ whose H_{sub}-neighborhood type is w.

The extensions can be counted by a straightforward recursive algorithm provided that the numbers t_w and $n_b(\alpha)$ are known. If H has $O(1)$ vertices then the algorithm runs in $O(1)$ time. See the full version for details. Also, the numbers t_w can be computed in $O(1)$ time if H has $O(1)$ vertices.

Note that for two such distinct l-tuples α, the subgraph isomorphisms f_α between H_{sub} and G are different and thus their extensions to a subgraph isomorphism between H and G have to be different too. Thus, it is sufficient to sum the numbers of extensions over such l-tuples α.

We can list all such l-tuples α where the function f_α mapping the i-th vertex of H_{sub} on the i-th vertex of α is a subgraph isomorphism between H_{sub} and G in $O(n^l)$ time if $l = O(1)$. Finally, all the numbers $n_w(\alpha)$ can be obtained by solving the l-neighborhood problem in time $O(2^l n^{\omega(\lceil l/2\rceil,1,\lfloor l/2\rfloor)})$ by Fact 1. □

Suppose that the edges and/or vertices of the host graph G have some real weights. For the problem of counting lightest subgraph isomorphisms between the pattern graph H and G, i.e., the isomorphisms between H and lightest subgraphs isomorphic to H, we obtain the following theorem whose proof is given in the full version.

Theorem 6. *Let H be a pattern graph on k vertices with an independent set on s vertices, and let G be a host graph on n vertices with vertex and/or edge real weights. If $k = O(1)$ then the number of lightest subgraph isomorphisms between H and G can be computed by a combinatorial algorithm in $O(n^{k-s+1} \log n)$ time.*

References

1. Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., Sahinalp, S.C.: Biomolecular network motif counting and discovery by color coding. Bioinformatics (ISMB 2008) 24(13), 241–249 (2008)
2. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: Proc. of FOCS 2010, pp. 173–182 (2010)
3. Corneil, D.G., Perl, Y., Stewart, L.K.: A Linear Recognition Algorithm for Cographs. SIAM J. Comput. 14(4), 926–934 (1985)
4. Czumaj, A., Lingas, A.: Finding a Heaviest Vertex-Weighted Triangle is not Harder than Matrix Multiplication. SIAM J. Comput. 39(2), 431–444 (2009)
5. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Information Processing Letters 7, 193–195 (1978)
6. Edmonds, J.: Systems of distinct representatives and linear algebra. J. Res. Nat. Bur. Standards Sect. B 7B1, 241–245 (1967)
7. Eisenbrand, F., Grandoni, F.: On the complexity of fixed parameter clique and dominating set. Theoretical Computer Science 326(1:3), 57–67 (2004)
8. Floderus, P., Kowaluk, M., Lingas, A., Lundell, E.-M.: Induced subgraph isomorphism: Are some patterns substantially easier than others? In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 37–48. Springer, Heidelberg (2012)
9. Fomin, F.V., Lokshtanov, D., Raman, V., Rao, B.V.R., Saurabh, S.: Faster Algorithms for Finding and Counting Subgraphs. J. Comput. and Syst. Sci. 78(3), 698–706 (2012)
10. Gelbord, B.: Graphical techniques in intrusion detection systems. In: Proceedings 15th International Conference on Information Networks, pp. 253–258 (2001)
11. Hoang, C.T., Kaminski, M., Sawada, J., Sritharan, R.: Finding and listing induced paths and cycles. Discrete Applied Mathematics 161(4-5), 633–641 (2013)
12. Kloks, T., Kratsch, D., Müller, H.: Finding and counting small induced subgraphs efficiently. Information Processing Letters 74(3-4), 115–121 (2000)
13. Kowaluk, M., Lingas, A., Lundell, E.M.: Counting and detecting small subgraphs via equations and matrix multiplication. SIAM J. Discrete Math. 27(2), 892–909 (2013)
14. Koutis, I., Williams, R.: Limits and Applications of Group Algebras for Parameterized Problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
15. Le Gall, F.: Faster Algorithms for Rectangular Matrix Multiplication. In: Proc. of FOCS, pp. 514–523 (2012)
16. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae 26(2), 415–419 (1985)
17. Olariu, S.: Paw-Free Graphs. Information Processing Letters 28, 53–54 (1988)
18. Schank, T., Wagner, D.: Finding, counting and listing all triangles in large graphs, an experimental study. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 606–609. Springer, Heidelberg (2005)
19. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM 27, 701–717 (1980)
20. Vassilevska Williams, V.: Multiplying matrices faster than coppersmith-winograd. In: Proc. of STOC, pp. 887–898 (2012)
21. Vassilevska, V.: Efficient Algorithms for Path Problems in Weighted Graphs, PhD thesis, CMU, CMU-CS-08-147 (2008)
22. Vassilevska, V., Williams, R.: Finding, Minimizing, and Counting Weighted Subgraphs. In: Proc. of STOC 2009, pp. 455–464 (2009)

An $O^*(1.1939^n)$ Time Algorithm for Minimum Weighted Dominating Induced Matching

Min Chih Lin[1,⋆], Michel J. Mizrahi[1,⋆], and Jayme L. Szwarcfiter[2,⋆⋆]

[1] CONICET, Instituto de Cálculo and Departamento de Computación Universidad de Buenos Aires, Buenos Aires, Argentina

[2] I. Mat., COPPE and NCE, Universidade Federal do Rio de Janeiro, and Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brazil

oscarlin@dc.uba.ar, michel.mizrahi@gmail.com, jayme@nce.ufrj.br

Abstract. Say that an edge of a graph G dominates itself and every other edge sharing a vertex of it. An edge dominating set of a graph $G = (V, E)$ is a subset of edges $E' \subseteq E$ which dominates all edges of G. In particular, if every edge of G is dominated by exactly one edge of E' then E' is a dominating induced matching. It is known that not every graph admits a dominating induced matching, while the problem to decide if it does admit it is NP-complete. In this paper we consider the problems of finding a minimum weighted dominating induced matching, if any, and counting the number of dominating induced matchings of a graph with weighted edges. We describe an exact algorithm for general graphs that runs in $O^*(1.1939^n)$ time and polynomial (linear) space, for solving these problems. This improves over the existing exact algorithms for the problems in consideration.

Keywords: exact algorithms, dominating induced matchings, branch & reduce.

1 Introduction

Under the widely accepted assumption that $P \neq NP$ there are several problems with important applications for which no polynomial algorithm exists. The need to get an exact solution for many of those problems has lead to a growing interest in the area of design and analysis of exact exponential time algorithms for NP-Hard problems [14,25]. Even a slight improvement of the base of the exponential running time may increase the size of the instances being tractable. There has been many new and promising advances in recent years towards this direction [1,2].

⋆ Partially supported by UBACyT Grants 20020100100754 and 20020120100058, PICT ANPCyT Grant 1970 and PIP CONICET Grant 11220100100310.

⋆⋆ Partially supported by CNPq, CAPES and FAPERJ, research agencies.

In this paper we give an exact algorithm for the weighted and counting versions of the problem Dominating Induced Matching (also known as DIM or Efficient Edge Domination) which has been extensively studied [3,4,5,7,8,17,9,20,21]. Further notes about this problem and some applications related to encoding theory, network routing and resource allocation can be found in [15,19].

The unweighted version of the DIM problem is known to be NP-complete [15], even for planar bipartite graphs of maximum degree 3 [3] or regular graphs [9]. There are polynomial time algorithms for some classes, such as chordal graphs [20], generalized series-parallel graphs [20] (both for the weighted problem), claw-free graphs [7], graphs with bounded clique-width [7], hole-free graphs [3], convex graphs [17], dually-chordal graphs [4], P_7-free graphs [5], bipartite permutation graphs [21], AT-free graphs [4], interval-filament graphs [4], weakly chordal graphs [4]. See also [6].

If $P \neq NP$ it is not possible to solve this problem in polynomial time, hence it becomes important to improve the exponential algorithm in order to identify instances that can be solved within reasonable time.

A straightforward brute-force algorithm using an alternative definition of the problem explained later to solve the weighted DIM requires $O^*(2^n)$ time and polynomial space.

The paper [18] describes an algorithm for solving the weighted DIM problem in $O^*(1.7818^n)$, time while requiring $O(n + m)$ time, if the graph contains a vertex dominating set of fixed size. In the same work another approach based on enumerating maximal independent sets allows to solve both DIM problems (minimum weighted problem and counting problem) in $O^*(1.4423^n)$ time. There is also an $O^*(1.5849^n)$ algorithm from [16].

The minimum weighted DIM problem can be expressed as an instance of the maximum weighted independent set (MWIS) problem on the square of the line graph $L(G)$ of G, and also as an instance of the minimum weighted dominating set problem on $L(G)$, by slightly adjusting the models [4,22] for the unweighted DIM problem. The MWIS problem can be solved in $O^*(1.2377^n)$ time [24] (how one obtains an algorithm for MWIS from an algorithm for weighted 2-Sat is described in [10]). On the other hand, the minimum weighted dominating set problem can be solved in time $O^*(1.5535^n)$ [12], and the special case where the weights are polynomially bounded in time $O^*(1.5048^n)$ [23]. Hence the minimum weighted DIM problem for a graph G can be solved by using the $L^2(G)$ algorithm in $O^*(1.2377^m)$ time using the MWIS algorithm and in $O^*(1.5048^m)$ time using the minimum weighted dominating set algorithm.

For the counting problem, there exist algorithms such as [11] which can be used to count the number of MWIS's in $O^*(1.3247^n)$ time, leading an $O^*(1.3247^m)$ time algorithm to count the numbers of DIM's.

In this paper, we propose an algorithm for solving the problems of finding the minimum weighted DIM and that of counting the DIM's in $O(m \cdot 1.1939^n) \in O^*(1.1939^n)$ time and $O(m)$ space in general graphs, which improves over the existing algorithms. We employ techniques described in [14] for the analysis of our algorithm, and as such we use their terminology. The proposed algorithm was designed using the *branch & reduce paradigm*. More information about this design technique as well as the running time analysis for this kind of algorithms can be found in [14].

2 Preliminaries

By $G(V, E)$ we denote a *simple undirected graph* with vertex set V and edge set E, $n = |V|$ and $m = |E|$. We consider G as a *weighted* graph, that is, one in which there is a non-negative real value, denoted $weight(vw)$ assigned to each edge vw of G. If $v \in V$ and $V' \subseteq V$, then denote by $N(v)$, the set of vertices adjacent (neighbors) to v, denote $d(v) = |N(v)|$ the degree of the vertex, denote by $G[V']$ the subgraph of G induced by V', and write $N_{V'}(v) = N(v) \cap V'$. Some special graphs or vertices are of interest for our purposes. A graph formed by two triangles having a common edge is called a *diamond*. By removing an edge incident to a vertex of degree 2 of a diamond, we obtain a *paw*. Finally, a vertex of degree 1 is called *pendant*.

Given an edge $e \in E$, say that *e dominates* itself and every edge sharing a vertex with e. Subset $E' \subseteq E$ is an *induced matching* of G if each edge of G is dominated by at most one edge in E'. A *dominating induced matching (DIM)* of G is a subset of edges which is both dominating and an induced matching. Not every graph admits a DIM, and the problem of determining whether a graph admits it is also known in the literature as *efficient edge domination problem*. The weighted version of DIM problem is to find a DIM such that the sum of the weights of its edges is minimum among all DIM's, if any. The counting version of the problem consists on counting the number of DIM's of the graph. We assume the weights to be non-negative. However, the methods can be easily extended to the case of negative weights, without increasing the complexity of the algorithms.

It is not hard to see that every DIM is a maximum induced matching, and hence the number of edges of every DIM in G is the same. Therefore it is straightforward to modify the graph in order to solve the problem with non-negative weights and then transform it back to the original graph.

We assume the graph G to be connected, otherwise, the DIM of G is the union of the DIM's of its connected components, and so we can restrict to the connected case.

We will use an alternative definition [8] of the problem of finding a dominating induced matching. It asks to determine if the vertex set of a graph G admits a partition into two subsets. The vertices of the first subset are called *white* and induce an independent set of the graph, while those of the second subset are named *black* and induce an 1-regular graph.

A straightforward brute-force algorithm for finding the DIM of a graph G consists in finding all bipartitions of $V(G)$, color one of the parts as *white*, the other as *black*, and checking if the result is a valid DIM. The complexity of this algorithm is $O(2^n \cdot m) \in O^*(2^n)$.

3 Extensions of Colorings

Assigning one of the two possible colors, white or black, to vertices of G is called a coloring of G. A coloring is *partial* if only part of the vertices of G have been assigned colors, otherwise it is *total*. A black vertex is *single* if it has no black neighbor, and is paired if it has exactly one black neighbor. Each coloring, partial or total, can be *valid* or *invalid*.

Next, we describe the natural conditions for determining if a coloring is valid or invalid.

Definition 1. *RULES FOR VALIDATING COLORINGS:*

A partial coloring is valid whenever:

(V1). *No two white vertices are adjacent, and*
(V2). *Each black vertex is either single or paired. Each single vertex has some uncolored neighbor.*

A total coloring is valid whenever:

(V3). *No two white vertices are adjacent, and*
(V4). *Each black vertex is paired.*

Lemma 1. *There is a one-to-one correspondence between total valid colorings and dominating induced matchings of a graph.*

Proof: It follows from the definitions. □

For a coloring C of the vertices of G, denote by $C^{-1}(white)$ and $C^{-1}(black)$, the subsets of vertices colored white and black. A coloring C' is an *extension* of a C if $C^{-1}(black) \subseteq C'^{-1}(black)$ and $C^{-1}(white) \subseteq C'^{-1}(white)$. For $V', V'' \subset V(G)$ if C' is obtained from C by adding to it the vertices of V' with the color black and those of V'' with the color white then write $C = C' \cup BLACK(V') \cup WHITE(V'')$.

Given a partial coloring C, the basic idea of the algorithm is to iteratively find extensions C' of C, until eventually a total valid coloring is reached. It follows from the validation rules that if C is invalid, so is C'. Therefore, the algorithm keeps checking for validation, and would discard an extension whenever it becomes invalid.

Basically, there are two different ways of possibly extending a coloring, using propagation rules and branching rules. At the beginning, there are partial colorings C which force the colors of some of the so far uncolored vertices, leading to an extension C' of C. In this case, say that C' has been obtained from C by *propagation*. The following is a convenient set of rules, whose application may extend C, in the above described way.

Lemma 2. *RULES FOR PROPAGATING COLORS:*
The following are forced colorings for the extensions of a partial coloring of G.

(P1). *The degree-3 vertices of a diamond must be black and the remaining ones must be white*
(P2). *The neighbor of a pendant vertex must be black*
(P3). *Each neighbor of a white vertex must be black*
(P4). *Except for its pair, the neighbors of a paired (black) vertex must be white*
(P5). *Each vertex with two black neighbors must be white*
(P6). *If a single black vertex has exactly one uncolored neighbor then this neighbor must be black*
(P7). *In an induced paw, the two odd-degree vertices must have different colors*
(P8). *In an induced C_4, adjacent vertices must have different colors*
(P9) *If $\forall v \in N_U(s), N[v] \subseteq N[s]$ where s is a single (black vertex) then an uncolored neighbor v of s minimizing weight(sv) must be black. Break ties arbitrarily. We require rules (P1). and (P8). to be applied before (P9).*

Lemma 3. *[3] If G contains a K_4 then G has no DIM.*

Say that a coloring C is *empty* if all vertices are uncolored. Let C be a valid coloring and C' an extension of it, obtained by the application of the propagation rules. If $C = C'$ then C is called *stable.* On the other hand, if $C \neq C'$ then C' is not necessarily valid. Therefore, after applying iteratively the propagation rules, we reach an extension which is either invalid or stable. In order to possibly extend a stable coloring C, we apply *branching rules.* Any coloring directly obtained by these rules is not forced. Instead, in each of the these rules, there are two possibly conflicting alternatives leading to distinct extensions C'_1, C'_2 of C. Each of C'_1 or C'_2 may be independently valid or invalid. The next lemma describes the branching rules. We remark that there exist simpler branching rules. However, using the rules below we obtain a sufficient number of vertices that get forced colorings, through the propagation which follow the application of any branching rule, so as to guarantee a decrease of the overall complexity of the algorithm. The complexity obtained relies heavily on this fact.
In general, we adopt the following notation. If C is a stable coloring then S denotes the set of single vertices of it , U is the set of uncolored vertices and $T = U \setminus \cup_{s \in S} N_U(s)$.

Lemma 4. *BRANCHING RULES*
Let C be a partial (valid) stable coloring of a graph G. At least one of the following alternatives can be applied to define extensions C'_1, C'_2 of C.

(B1) *If C is an empty coloring: choose an arbitrary vertex v then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*

(B2) *If $\exists$ edge vw s.t. $v \in N_U(s)$ and $w \in N_U(s')$, for some $s, s' \in S, s \neq s'$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*

(B3) *For some $s \in S$, if $\exists v \in N_U(s)$ s.t. $\exists w \in N_T(v)$:*

B3(a) *If $|N_U(s)| \neq 3 \vee d(w) \neq 3 \vee |N_T(v)| \geq 2$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$.*

B3(b) *If $|N_U(s)| = 3 \wedge d(w) = 3 \wedge N_T(v) = \{w\}$, let $N_U(s) = \{v, v', v''\}$.*

B3(b).i *If $N_U(v') = N_U(v'') = \emptyset$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*

B3(b).ii *If $N_U(v') \neq \emptyset$, let $w' \in N_T(v')$, with $w' \neq w$. If $|N(w) \cup N(w')| > 5$ or $ww' \notin E(G)$ then $C'_1 := C \cup BLACK(\{v\})$ and $C'_2 := C \cup WHITE(\{v\})$*

B3(b).iii *If $N_U(v') \neq \emptyset$, let $w' \in N_T(v')$, with $w' \neq w$. If $ww' \in E(G)$ and $z \in N(w) \cap N(w')$ then $C'_1 := C \cup BLACK(\{v''\})$, while if $weight(sv) + weight(w'z) \leq weight(sv') + weight(wz)$ then $C'_2 := C \cup BLACK(\{v\})$, otherwise $C'_2 := C \cup BLACK(\{v'\})$*

Each rule is applied after the previous rule, that is, if the condition of the previous case is not verified in the entire graph. Note that this applies to subitems of case (B3).

4 The Algorithm

The lemmas described in the last section lead to an exact algorithm for finding a minimum weight DIM of a graph G, if any.

In the initial step of the algorithm, we find the set $K4$ containing the K_4's of G. If $K4 \neq \emptyset$, by Lemma 3, G does not have DIM's, and terminate the algorithm. Otherwise, define the set $COLORINGS$ to contain through the process the candidates colorings to be examined and eventually extended. Next, include in COLORINGS an *empty* coloring. In the general step, we choose any coloring C from $COLORINGS$ and remove it from this set. Then iteratively propagate the coloring by Lemma 2 into an extension C' of it, and validate the extension by Definition **1** The iterations are repeated until one of the following situations is reached: C' is invalid, C' is a total valid coloring, or a partial stable (valid) coloring. In the first alternative, C' is discarded and a new coloring from $COLORINGS$ is chosen. If C' is a a total valid coloring, find its weight and if smaller than the least weight so far obtained, it becomes the current candidate for the minimum weight of a DIM of G. Finally, when C' is stable we extended it by branching rules: choose the first rule of Lemma 4 satisfying C', compute the extensions C' and C'', insert them in $COLORINGS$, select a new coloring from $COLORINGS$ and repeat the process.

The formulation below describes the details. The propagation and validation of a coloring C are performed by the procedure $PROPAGATE$ –

$VALIDATE(C, RESULT)$. At the end, the returned coloring corresponds to the extension C' of C, after iteratively applying propagation. The variable RESULT indicates the outcome of the validation analysis. If C' is invalid then $RESULT$ is 'invalid'; if C' is a valid total coloring then it contains 'total', and otherwise $RESULT$ equals 'partial'. Finally, $BIFURCATE(C, C'_1, C'_2)$ computes the extensions C'_1 and C'_2 of C.

Algorithm Minimum Weighted DIM / Counting DIM

```
1. Find the subset $K4$
   if $K4 \neq \emptyset$ then terminate the algorithm: $G$ has no DIM
       $SOLUTION := NODIM$
2. $COLORINGS := \{C\}$, where $C$ is an empty coloring
3. while $COLORINGS \neq \emptyset$ do
   a.   choose $C \in COLORINGS$ and remove it from $COLORINGS$
   b.   $PROPAGATE - VALIDATE(C, RESULT)$
   c.   if $RESULT$ = 'total' and $weight(C) < SOLUTION$ then
            $SOLUTION := weight(C)$
        else if $RESULT$ = 'partial' then
            Set $C'_1$ and $C'_2$ according to branching RULES on $C$
            $COLORINGS := COLORINGS \cup \{C'_1, C'_2\}$
        end if
4. Output $SOLUTION$
```

procedure $PROPAGATE - VALIDATE(C, RESULT)$

```
Comment Phase 1: Propagation
1. $C' := C$
2. repeat
   $C := C'$
   $C'$ := extension of $C$ obtained by the PROPAGATION RULES until
   $C = C'$
Comment Phase 2: Validation
3. Using the VALIDATION RULES 1 do as follows:
   if $C$ is an invalid coloring then return ($C$, 'invalid')
   else if $C$ is a partial coloring then return ($C$,'partial')
   else return ($C$, 'total')
```

5 Correctness and Complexity

It is easy to see that our algorithm fits the *branch & reduce* paradigm [14]. The *propagation rules* can be mapped into *reduction rules.*

Theorem 1. *The algorithm described in the previous section correctly computes the minimum weight of a dominating induced matching of a graph G.*

Proof: The correctness of the algorithm follows from the fact that the algorithm considers all colorings that represent a DIM that can have minimum weight. Lemmas 2 and 4 are applied to extend partial colorings. Invalid colorings are discarded, while valid colorings are further extended, except if some other valid coloring representing a better DIM (with less weight) appeared before.

For proving the worst-case running time upperbound for the algorithm we will use the following useful definition and theorem.

Definition 2. *[14] Let b a branching rule and n the size of the instance. Suppose rule b branches the current instance into $r \geq 2$ instances of sizes respectively at most $n-t_1, n-t_2, \ldots, n-t_r$, for all instances of size $n \geq max\{t_i : i = 1, 2, \ldots, r\}$. Then we call $b = (t_1, t_2, \ldots, t_r)$ the branching vector of branching rule b.*

The branching vector $b = (t_1, t_2, \ldots, t_r)$ implies the linear recurrence $T(n) \leq T(n-t_1) + T(n-t_2) + \ldots, T(n-t_r)$.

Theorem 2. *[14] Let b be a branching rule corresponding to the branching vector $(t_1, t_2, \ldots, t_r)$. Then the running time of the branching algorithm using only branching rule b is $O^*(\alpha^n)$, where α is the unique positive real root of*

$$x^n - x^{n-t_1} - x^{n-t_2} - \ldots - x^{n-t_r} = 0$$

The unique positive real root α is the *branching factor* of the branching vector b. We denote the branching factor of $(t_1, t_2, \ldots, t_r)$ by $\tau(t_1, t_2, \ldots, t_r)$.

Therefore for analyzing the running time of a branching algorithm we can compute the factor α_i for every branch rule b_i, and an upper bound of the running time of the branching algorithm is obtained by taking $\alpha = max_i \alpha_i$ and the result is an upper bound for the running time of $O^*(\alpha^n)$.

The upper bound is obtained by counting the leaves of the search tree given by the algorithm, using the fact that each leaf can be executed in polynomial time. The complexity of the algorithm without hiding the polynomial factor depends on the time for the execution of each branch in the search tree.

Further notes on this topic can be found in [14]

Theorem 3. *The algorithm above described requires $O^*(1.1939^n)$ time and $O(n+m)$ space for completion.*

6 Counting the Number of DIM's

The previous algorithm can be easily adapted to count the number of DIM's. Given a coloring C we define $TVC(C)$ the number of *total valid* colorings that can be extended from C. If we apply any propagation rule to coloring C we obtain a coloring C'. Clearly $TVC(C) = TVC(C')$, except for rule (P9). In the latter case $TVC(C) = TVC(C') \cdot |N_U(s)|$ where s is the single vertex chosen to apply the rule.

If we apply any branching rule to coloring C we obtain two extended colorings C_1' and C_2'. Clearly $TVC(C) = TVC(C_1') + TVC(C_2')$, except for rule B3(b).iii. In the latter case $TVC(C) = TVC(C_1') + 2 \cdot TVC(C_2')$.

Using the above facts it is trivial to modify the algorithm to solve the counting problem.

References

1. Bjorklund, A.: Determinant sums for undirected hamiltonicity. In: Annual Symposium on Foundations of Computer Science, FOCS 2010, pp. 173–182 (2010)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets möbius: fast subset convolution. In: Annual Symposium on Foundations of Computer Science, STOC 2007, pp. 67–74 (2007)
3. Brandstädt, A., Hundt, C., Nevries, R.: Efficient edge domination on hole-free graphs in polynomial time. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 650–661. Springer, Heidelberg (2010)
4. Brandstädt, A., Leitert, A., Rautenbach, D.: Efficient dominating and edge dominating sets for graphs and hypergraphs. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 267–277. Springer, Heidelberg (2012)
5. Brandstädt, A., Mosca, R.: Dominating Induced Matchings for P_7-free Graphs in Linear Time. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 100–109. Springer, Heidelberg (2011)
6. Brandstädt, A., Lozin, V.V.: On the linear structure and clique-width of bipartite permutation graphs. Ars Combinatoria 67, 273–281 (2003)
7. Cardoso, D.M., Korpelainen, N., Lozin, V.V.: On the complexity of the dominating induced matching problem in hereditary classes of graphs. Discrete Applied Mathematics 159, 21–531 (2011)
8. Cardoso, D.M., Lozin, V.V.: Dominating induced matchings. In: Lipshteyn, M., Levit, V.E., McConnell, R.M. (eds.) Graph Theory, Computational Intelligence and Thought. LNCS, vol. 5420, pp. 77–86. Springer, Heidelberg (2009)
9. Cardoso, D.M., Cerdeira, J.O., Delorme, C., Silva, P.C.: Efficient edge domination in regular graphs. Discrete Applied Mathematics 156, 3060–3065 (2008)
10. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2sat and 3sat formulae. Theoretical Computer Science 332, 265–291 (2005)
11. Dahllöf, V., Jonsson, P.: An algorithm for counting maximum weighted independent sets and its applications. In: ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, pp. 292–298 (2002)
12. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica 54, 181–207 (2009)
13. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O^*(1.220^n)$ independent set algorithm. In: SODA 2006 ACM-SIAM Symposium on Discrete Algorithms, pp. 18–25 (2006)
14. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. EATCS Series in Theoretical Computer Science. Springer, Berlin (2010)
15. Grinstead, D.L., Slater, P.J., Sherwani, N.A., Holmes, N.D.: Efficient edge domination problems in graphs. Information Processing Letters 48, 221–228 (1993)
16. Gupta, S., Raman, V., Saurabh, S.: Maximum r-regular induced subgraph problem: Fast exponential algorithms and combinatorial bounds. SIAM Journal on Discrete Mathematics 26, 1758–1780 (2012)
17. Korpelainen, N.: A polynomial-time algorithm for the dominating induced matching problem in the class of convex graphs. Electronic Notes in Discrete Mathematics 32, 133–140 (2009)
18. Lin, M.C., Mizrahi, M.J., Szwarcfiter, J.L.: Exact algorithms for dominating induced matchings. CoRR, abs/1301.7602 (2013)

19. Livingston, M., Stout, Q.F.: Distributing resources in hypercube computers. In: C3P Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer Systems, and General Issues, vol. 1, pp. 222–231. ACM (1988)
20. Lu, C.L., Ko, M.-T., Tang, C.Y.: Perfect edge domination and efficient edge domination in graphs. Discrete Applied Mathematics 119, 227–250 (2002)
21. Lu, C.L., Tang, C.Y.: Solving the weighted efficient edge domination problem on bipartite permutation graphs. Discrete Applied Mathematics 87, 203–211 (1998)
22. Milanic, M.: Hereditary efficiently dominatable graphs. Journal of Graph Theory 73, 400–424 (2013)
23. van Rooij, J.M.M., Nederlof, J., van Dijk, T.C.: Inclusion/exclusion meets measure and conquer. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 554–565. Springer, Heidelberg (2009)
24. Wahlström, M.: A tighter bound for counting max-weight solutions to 2SAT instances. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 202–213. Springer, Heidelberg (2008)
25. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Combinatorial Optimization - Eureka, you Shrink!, pp. 185–207. Springer-Verlag New York, Inc., New York (2003)

New Inapproximability Bounds for TSP[*]

Marek Karpinski[1,**], Michael Lampis[2,***], and Richard Schmied[3,†]

[1] Dept. of Computer Science and the Hausdorff Center for Mathematics, University of Bonn
marek@cs.uni-bonn.de

[2] Research Institute for Mathematical Sciences (RIMS), Kyoto University
mlampis@kurims.kyoto-u.ac.jp

[3] Dept. of Computer Science, University of Bonn
schmied@cs.uni-bonn.de

Abstract. In this paper, we study the approximability of the metric Traveling Salesman Problem (TSP) and prove new explicit inapproximability bounds for that problem. The best up to now known hardness of approximation bounds were 185/184 for the symmetric case (due to Lampis) and 117/116 for the asymmetric case (due to Papadimitriou and Vempala). We construct here two new bounded occurrence CSP reductions which improve these bounds to 123/122 and 75/74, respectively. The latter bound is the first improvement in more than a decade for the case of the asymmetric TSP. One of our main tools, which may be of independent interest, is a new construction of a bounded degree wheel amplifier used in the proof of our results.

1 Introduction

The **Traveling Salesman Problem (TSP)** is one of the best known and most fundamental problems in combinatorial optimization. Determining how well it can be approximated in polynomial time is therefore a major open problem, albeit one for which the solution still seems elusive. On the algorithmic side, the best known efficient approximation algorithm for the symmetric case is still a 35-year old algorithm due to Christofides [9] which achieves an approximation ratio of 3/2. However, recently there has been a string of improved results for the interesting special case of Graphic TSP, improving the ratio to 7/5 [18,16,17,21]. For the asymmetric case (ATSP), it is not yet known if a constant-factor approximation is even possible, with the best known algorithm achieving a ratio of $O(\log n/ \log\log n)$ [2].

[*] The full version of the paper is to be found under: http://arxiv.org/abs/1303.6437

[**] Supported in part by DFG grants and the Hausdorff Grant EXC59-1/2.

[***] Research partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan. Part of this work was done while the second author was at KTH Royal Institute of Technology, supported by ERC Grant 226203.

[†] Work supported by Hausdorff Doctoral Fellowship.

Unfortunately, there is still a huge gap between the algorithmic results mentioned above and the best currently known hardness of approximation results for TSP and ATSP. For both problems, the known inapproximability thresholds are small constants (185/184 and 117/116 (cf. [15,19]), respectively). In this paper, we try to improve this situation somehow by giving modular hardness reductions that slightly improve the hardness bounds for both problems to 123/122 and 75/74, respectively. The latter bound is the first, for more than a decade now, improvement of Papadimitriou and Vempala bound [19] for the ATSP. The method of our solution differs essentially from that of [19] and uses some new paradigms of the bounded occurrence optimization which could be also of independent interest in other applications. Similarly to [15], the hope is that the modularity of our construction, which goes through an intermediate stage of a bounded-occurrence Constraint Satisfaction Problem (CSP), will allow an easier analysis and simplify future improvements. Indeed, one of the main new ideas we rely on is a certain new variation of the wheel amplifiers first defined by Berman and Karpinski [4] to establish inapproximability for 3-regular CSPs. This construction, which may be of independent interest, allows us to establish inapproximability for a 3-regular CSP with a special structure. This special structure then makes it possible to simulate many of the constraints in the produced graph essentially "for free", without using gadgets to represent them. Thus, even though for the remaining constraints we mostly reuse gadgets which have already appeared in the literature, we are still able to obtain improved bounds.

Let us now recall some of the previous work on the hardness of approximation of TSP and ATSP. Papadimitriou and Yannakakis [20] were the first to construct a reduction that, combined with the PCP Theorem [1], gave a constant inapproximability threshold, though the constant was not more than $1+10^{-6}$ for the TSP with distances either one or two. Engebretsen [10] gave the first explicit approximation lower bound of 5381/5380 for the problem. The inapproximability factor was improved to 3813/3812 by Böckenhauer and Seibert [8], who studied the restricted version of the TSP with distances one, two and three. Papadimitriou and Vempala [19] proved that it is **NP**-hard to approximate the TSP with a factor better than 220/219. Presently, the best known approximation lower bound is 185/184 due to Lampis [15].

The important restriction of the TSP, in which we consider instances with distances between cities being values in $\{1, \ldots, B\}$, is often referred to as the $(1, B)$-TSP. The best known efficient approximation algorithm for the $(1, 2)$-TSP has an approximation ratio 8/7 and is due to Berman and Karpinski [6]. As for lower bounds, Engebretsen and Karpinski [11] gave inapproximability thresholds for the $(1, B)$-TSP problem of 741/740 for $B = 2$ and 389/388 for $B = 8$. More recently, Karpinski and Schmied [13,14] obtained improved inapproximability factors for the $(1, 2)$-TSP and the $(1, 4)$-TSP of 535/534 and 337/336, respectively.

For ATSP the currently best known approximation lower bound was 117/116 due to Papadimitriou and Vempala [19]. When we restrict the problem to distances with values in $\{1, \ldots, B\}$, there is a simple approximation algorithm with approximation ratio B that constructs an arbitrary tour as solution. Bläser [7] gave an efficient approximation algorithm for the $(1, 2)$-ATSP with approximation ratio 5/4. Karpinski and Schmied [13,14] proved that it is **NP**-hard to approximate the $(1, 2)$-ATSP and the $(1, 4)$-ATSP within any factor less than 207/206 and 141/140, respectively. For the case $B = 8$, Engebretsen and Karpinski [11] gave an inapproximability bound of 135/134.

Overview: In this paper we give a hardness proof which proceeds in two steps. First, we start from the MAX-E3-LIN2 problem, in which we are given a system of linear equations mod 2 with exactly three variables in each equation and we want to find an assignment such as to maximize the number of satisfied equations. Optimal inapproximability results for this problem were shown by Håstad [12]. We reduce this problem to a special case where variables appear exactly 3 times and the linear equations have a particular structure. The main tool here is a new variant of the wheel amplifier graphs of Berman and Karpinski [4].

In the second step, we reduce this 3-regular CSP to TSP and ATSP. The general construction is similar in both cases, though of course we use different gadgets for the two problems. The gadgets we use are mostly variations of gadgets which have already appeared in previous reductions. Nevertheless, we manage to obtain an improvement by exploiting the special properties of the 3-regular CSP. In particular, we show that it is only necessary to construct gadgets for roughly one third of the constraints of the CSP instance, while the remaining constraints are simulated without additional cost using the consistency properties of our gadgets. This idea may be useful in improving the efficiency of approximation-hardness reductions for other problems.

Thus, overall we follow an approach unlike that of [19], where the reduction is performed in one step, and closer to [15]. The improvement over [15] comes mainly from the idea mentioned above, which is made possible using the new wheel amplifiers, as well as several other tweaks. The end result is a more economical reduction which improves the bounds for both TSP and ATSP. An interesting question may be whether our techniques can also be used to derive improved inapproximability results for variants of the ATSP and TSP (cf. [11],[14] and [13]) or other graph problems, such as the Steiner Tree problem.

2 Preliminaries

In the following, we give some definitions concerning directed (multi-)graphs and omit the corresponding definitions for undirected (multi-)graphs if they follow from the directed case. Given a directed graph $G = (V(G), E(G))$ and $E' \subseteq E(G)$, for $e = (x, y) \in E(G)$, we define $V(e) = \{x, y\}$ and $V(E') = \bigcup_{e \in E'} V(e)$. For convenience, we abbreviate a sequence of edges (x_1, x_2), $(x_2, x_3), \ldots, (x_{n-1}, x_n)$ by $x_1 \to x_2 \to x_3 \to \ldots \to x_{n-1} \to x_n$. In the undirected case, we use sometimes $x_1 - x_2 - x_3 - \ldots - x_{n-1} - x_n$ instead of $\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{n-1}, x_n\}$. Given a directed (multi-)graph G, an *Eulerian* cycle in G is a directed cycle that traverses all edges of G exactly once. We refer to G as *Eulerian*, if there exists an Eulerian cycle in G. For a multiset E_T of directed edges and $v \in V(E_T)$, we define the outdegree (indegree) of v with respect to E_T, denoted by $outd_T(v)$ ($ind_T(v)$), to be the number of edges in E_T that are outgoing of (incoming to) v. The *balance* of a vertex v with respect to E_T is defined as $bal_T(v) = ind_T(v) - outd_T(v)$. In the case of a multiset E_T of undirected edges, we define the balance $bal_T(v)$ of a vertex $v \in V(E_T)$ to be one if the number of incident edges in E_T is odd and zero otherwise. We refer to vertices $v \in V(E_T)$ with $bal_T(v) = 0$ as *balanced* with respect to E_T. It is well known that a (directed) (multi-)graph $G = (V(G), E(G))$ is Eulerian if and only if all edges are in the same (weakly) connected component and all vertices $v \in V(G)$ are balanced with respect to $E(G)$.

Given a multiset of edges E_T, we denote by con_T the number of (weakly) connected components in the graph induced by E_T. A *quasi-tour* E_T in a (directed) graph G is a multiset of edges from $E(G)$ such that all vertices are balanced with respect to E_T and $V(E_T) = V(G)$. We refer to a quasi-tour E_T in G as a *tour* if $con_T = 1$. Given a cost function $w : E(G) \to \mathbb{R}_+$, the cost of a quasi-tour E_T in G is defined by $\sum_{e \in E_T} w(e) + 2(con_T - 1)$.

In the Asymmetric Traveling Salesman problem (ATSP), we are given a directed graph $G = (V(G), E(G))$ with positive weights on edges and we want to find an ordering $v_1, \ldots, v_n$ of the vertices such as to minimize $d_G(v_n, v_1) + \sum_{i \in [n-1]} d_G(v_i, v_{i+1})$, where d_G denotes the shortest path distance in G.

In this paper, we will use the following equivalent reformulation of the ATSP: Given a directed graph G with weights on edges, we want to find a tour E_T in G, that is, a spanning connected multi-set of edges that balances all vertices, with minimum cost.

The metric Traveling Salesman problem (TSP) is the special case of the ATSP, in which instances are undirected graphs with positive weights on edges.

3 Bi-Wheel Amplifiers

In this section, we define the bi-wheel amplifier graphs which will be our main tool for proving hardness of approximation for a bounded occurrence CSP with some special properties. Bi-wheel amplifiers are a simple variation of the wheel amplifier graphs given in [4]. Let us first recall some definitions (see also [5]).

If G is an undirected graph and $X \subset V(G)$ a set of vertices, we say that G is a *Δ-regular amplifier* for X if the following two conditions hold:

(i) All vertices of X have degree $\Delta - 1$ and all vertices of $V(G) \backslash X$ have degree Δ.

(ii) For every non-empty subset $U \subset V(G)$, we have the condition that $|E(U, V(G) \backslash U)| \geq \min\{\, |U \cap X|, |(V(G) \backslash U) \cap X| \,\}$, where $E(U, V(G) \backslash U) = \{e \in E(G) \mid 1 = |U \cap e|\}$.

We refer to the set X as the set of *contact* vertices and to $V(G) \backslash X$ as the set of *checker* vertices. Amplifier graphs are useful in proving inapproximability for CSPs, in which every variable appears a bounded number of times. Here, we will rely on 3-regular amplifiers. A probabilistic argument for the existence of such graphs was given in [4], with the definition of wheel amplifiers.

A wheel amplifier with $2n$ contact vertices is constructed as follows: first construct a cycle on $14n$ vertices. Number the vertices $1, 2 \ldots, 14n$ and select uniformly at random a perfect matching of the vertices whose number is not a multiple of 7. The matched vertices will be our checker vertices, and the rest our contacts. It is easy to see that the degree requirements are satisfied.

Berman and Karpinski [4] gave a probabilistic argument to prove that with high probability the above construction indeed produces an amplifier graph, that is, all partitions of the sets of vertices give large cuts. Here, we will use a slight variation of this construction, called a bi-wheel.

A **bi-wheel amplifier** with $2n$ contact vertices is constructed as follows: first construct two disjoint cycles, each on $7n$ vertices and number the vertices of each $1, 2 \ldots, 7n$. The contacts will again be the vertices whose number is a multiple of 7, while the remaining vertices will be checkers. To complete the construction, select uniformly at random a perfect matching from the checkers of one cycle to the checkers of the other.

Intuitively, the reason that amplifiers are a suitable tool here is that, given a CSP instance, we can use a wheel amplifier to replace a variable that appears $2n$ times with $14n$ new variables (one for each wheel vertex) each of which appears 3 times. Each appearance of the original variable is represented by a contact vertex and for each edge of the wheel we add an equality constraint between the corresponding variables. We can then use the property that all partitions give large cuts to argue that in an optimal assignment all the new vertices take the same value.

Before we apply the construction, we have to prove that the bi-wheel amplifiers still have the desired amplification properties. The proof of the following theorem is given in the full version of this paper.

Theorem 1. *With high probability, bi-wheels are 3-regular amplifiers.*

4 Hybrid Problem

By using the bi-wheel amplifier from the previous section, we are going to prove hardness of approximation for a bounded occurrence CSP with very special properties. This particular CSP will be well-suited for constructing a reduction to the TSP given in the next section.

As the starting point of our reduction, we make use of the inapproximability result due to Håstad [12] for the MAX-E3LIN2 problem, which is defined as follows: Given a system I_1 of linear equations mod 2, in which each equation is of the form $x_i \oplus x_j \oplus x_k = b_{ijk}$ with $b_{ijk} \in \{0, 1\}$, we want to find an assignment to the variables of I_1 such as to maximize the number of satisfied equations.

Let I_1 be an instance of the MAX-E3LIN2 problem and $\{x_i\}_{i=1}^{\nu}$ the set of variables, that appear in I_1. We denote by $d(i)$ the number of appearances of x_i in I_1.

Theorem 2 (Håstad [12])
*For every $\epsilon > 0$, there exists a constant B_ϵ such that given an instance I_1 of the MAX-E3LIN2 problem with m equations and $\max_{i\in[\nu]} d(i) \leq B_\epsilon$, it is **NP**-hard to decide whether there is an assignment that leaves at most $\epsilon \cdot m$ equations unsatisfied, or all assignment leave at least $(0.5 - \epsilon)m$ equations unsatisfied.*

Similarly to the work by Berman and Karpinski [3] (see also [4] and [5]), we will reduce the number of occurrences of each variable to 3. For this, we will use our amplifier construction to create special instances of the Hybrid problem, which is defined as follows: Given a system I_2 of linear equations mod 2 with either three or two variables in each equation, we want to find an assignment such as to maximize the number of satisfied equations.

In particular, we will use the following theorem, whose proof appears in the full version of this paper.

Theorem 3. *For every constant $\epsilon > 0$ and $b \in \{0, 1\}$, there exist instances of the Hybrid problem with $31m$ equations such that: (i) Each variable occurs exactly three times. (ii) $21m$ equations are of the form $x \oplus y = 0$, $9m$ equations are of the form $x \oplus y = 1$ and m equations are of the form $x \oplus y \oplus z = b$. (iii) It is **NP**-hard to decide whether there is an assignment to the variables that leaves at most $\epsilon \cdot m$ equations unsatisfied, or every assignment to the variables leaves at least $(0.5 - \epsilon)m$ equations unsatisfied.*

5 TSP

This section is devoted to the proof of the following theorem.

Theorem 4. *It is **NP**-hard to approximate the TSP to within any constant approximation ratio less than $123/122$.*

Let us first sketch the high-level idea of the construction. Starting with an instance of the Hybrid problem, we will construct a graph, where gadgets represent the equations. We will design gadgets for equations of size three (Figure 1) and for equations of size two corresponding to *matching* edges of the bi-wheel (Figure 2). We will not construct gadgets for the cycle edges of the bi-wheel; instead, the connections between the matching edge gadgets will be sufficient to encode these extra constraints. This may seem counter-intuitive at first, but the idea here is that if the gadgets for the matching edges are used in a consistent way (that is, the tour enters and exits in the intended way) then it follows that the tour is using all edges corresponding to one wheel and none from the other. Thus, if we prove consistency for the matching edge gadgets, we implicitly get the cycle edges "for free". This observation, along with an improved gadget for size-three equations and the elimination of the variable part of the graph, are the main sources of improvement over the construction of [15].

The Construction: In order to ensure that some edges are to be used at least once in any valid tour, we apply the following simple trick that was already used in the work by Lampis [15]: Let e be an edge with weight w that we want to be traversed by every tour. We remove e and replace it with a path of L edges and $L-1$ newly created vertices each of degree two, where we think of L as a large constant. Each of the L edges has weight w/L and any tour that fails to traverse at least two newly created edges is not connected. Any tour that traverses all but one of those edges can be extended by adding two copies of the unused edge increasing the cost of the underlying tour by a negligible value. In summary, we may assume that our construction contains *forced* edges that need to be traversed at least once by any tour. If x and y are vertices, which are connected by a forced edge e, we write $\{x,y\}_F$ or simply $x -_F y$. In the following, we refer to unforced edges e with $w(e)=1$ as *simple*. All unforced edges in our construction will be simple.

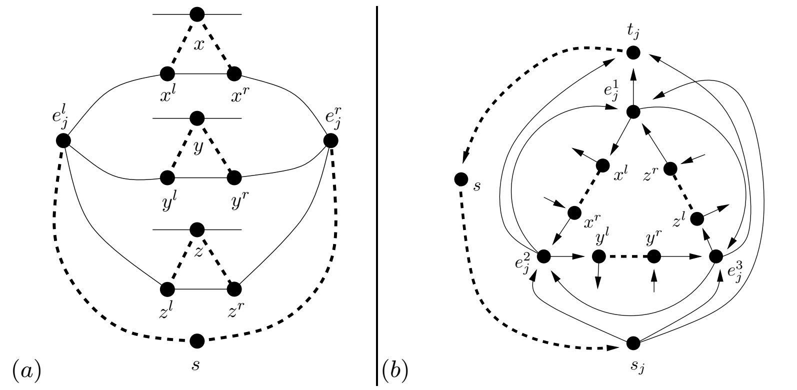


Fig. 1. Gadgets simulating equations with three variables in the symmetric case (a) and in the asymmetric case (b). Dotted and straight lines represent forced and simple edges, respectively.

Description of the corresponding graph G_S: For each bi-wheel W_p, we construct the subgraph G^p of G_S. For each vertex of the bi-wheel, we create a vertex in G^p and for each cycle equation $x \oplus y = 0$, we create a simple edge $\{x,y\}$. Given a matching equation $x_i^u \oplus x_j^n = 1$, we connect the vertices x_i^u and x_j^n via two forced edges $\{x_i^u, x_j^n\}_F^1$ and $\{x_i^u, x_j^n\}_F^2$ with $w(\{x_i^u, x_j^n\}_F^i) = 2$ for each $i \in \{1,2\}$. Additionally, we create a central vertex s that is connected to gadgets simulating equations with three variables. Due to Theorem 3, we may assume that equations with three variables in I_2 are all of the form $x \oplus y \oplus z = 0$. For the j-th equation with three variables in I_2, we now create the graph G_j^{3S} displayed in Figure 1 (a), where the (contact) vertices for x, y, z have already been constructed in the cycles. The edges $\{\gamma^\alpha, \gamma\}_F$ with $\alpha \in \{r,l\}$ and $\gamma \in \{x,z,y\}$ are all forced edges with $w(\{\gamma^\alpha,\gamma\}_F) = 1.5$. Furthermore, we have $w(\{e_j^\alpha, s\}_F) = 0.5$ for all $\alpha \in \{r,l\}$. $\{e_j^r, s\}_F$ and $\{e_j^l, s\}_F$ are both forced edges, whereas all remaining edges of G_j^{3S} are simple.

Tour from Assignment: Given an instance I_2 of the Hybrid problem and an assignment ϕ, we need to construct a tour in G_S according to ϕ. The proof of the following lemma is given in the full version of the paper.

Lemma 1. *If there is an assignment to the variables of a given instance I_2 of the Hybrid problem with $31m$ equations and ν bi-wheels, that leaves k equations unsatisfied, then, there exists a tour in G_S with cost at most $61m + 2\nu + k + 2$.*

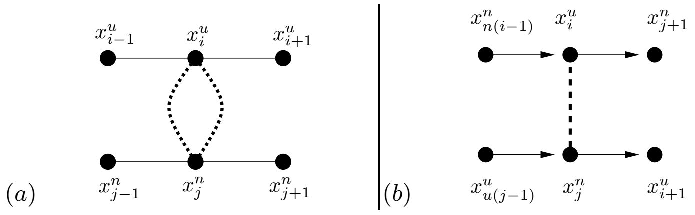


Fig. 2. Gadget simulating equation with two variables in symmetric case (a) and in the asymmetric case (b). Dotted and straight lines represent forced and simple edges, respectively.

Assignment from Tour: We now need to prove the other direction of our reduction. Given a tour in G_S, we must define an assignment to the variables of the associated instance of the Hybrid problem and prove the following lemma.

Lemma 2. *If there is a tour in G_S with cost $61m + k - 2$, then, there is an assignment to the variables of the corresponding instance of the Hybrid problem that leaves at most k equations unsatisfied.*

The proof of Lemma 2 is given in the full version of this paper. The proof of Theorem 4 follows from Lemmata 1, 2. Details are given in the full version of this paper.

6 ATSP

In this section, we prove the following theorem.

Theorem 5. *It is **NP**-hard to approximate the ATSP to within any constant approximation ratio less than $75/74$.*

Construction: Let us describe the construction that encodes an instance I_2 of the Hybrid problem into the instance G_A of the ATSP. Again, it will be useful to have the ability to force some edges to be used, that is, we would like to have bidirected forced edges. A bidirected forced edge between two vertices will be created in a similar way as undirected forced edges in the previous section. Without loss of generality, we may assume that bidirected forced edges are used in at least one direction, though we should note that the direction is not prescribed. In the remainder, we denote a directed forced edge consisting of vertices x and y by $(x, y)_F$, or $x \to_F y$.

Let I_2 consist of the collection $\{W_i\}_{i=1}^{\nu}$ of bi-wheels. Recall that the bi-wheel consists of two cycles and a perfect matching between their checkers. Let $\{x_i^u, x_i^n\}_{i=1}^{z}$ be the associated set of variables of W_p. We write $u(i)$ to denote the function which, given the index of a checker variable x_i^u returns the index j of the checker variable x_j^n to which it is matched (that is, the function u is a permutation function encoding the matching). We write $n(i)$ to denote the inverse function $u^{-1}(i)$.

Now, for each bi-wheel W_p, we are going to construct the corresponding directed graph G_A^p as follows. First, construct a vertex for each checker variable of the wheel. For each matching equation $x_i^u \oplus x_j^n = 1$, we create a bidirected forced edge $\{x_i^u, x_j^n\}_F$ with $w(\{x_i^u, x_j^n\}_F) = 2$.

For each contact variable x_k, we create two corresponding vertices x_k^r and x_k^l, which are joined by the bidirected forced edge $\{x_k^r, x_k^l\}_F$ with $w(\{x_k^r, x_k^l\}_F) = 1$.

Next, we will construct two directed cycles C_u^p and C_n^p. Note that we are doing arithmetic on the cycle indices here, so the index $z + 1$ should be read as equal to 1. For C_u^p, for any two consecutive checker vertices x_i^u, x_{i+1}^u on the un-negated side of the bi-wheel, we add a simple directed edge $x_{u(i)}^n \to x_{i+1}^u$. If the checker x_i^u is followed by a contact x_{i+1}^u in the cycle, then we add two simple directed edges $x_{u(i)}^n \to x_{i+1}^{ur}$ and $x_{i+1}^{ul} \to x_{i+2}^u$. Observe that by traversing the simple edges we have just added, the forced matching edges in the direction $x_i^u \to_F x_{u(i)}^n$ and the forced contact edges for the un-negated part in the direction $x_i^{ur} \to_F x_i^{ul}$, we obtain a cycle that covers all checkers and all the contacts of the un-negated part.

We now add simple edges to create a second cycle C_n^p. This cycle will require using the forced matching edges in the opposite direction and, thus, truth assignments will be encoded by the direction of traversal of these edges. First, for any two consecutive checker vertices x_i^n, x_{i+1}^n on the un-negated side of the bi-wheel, we add the simple directed edge $x_{n(i)}^u \to x_{i+1}^n$. Then, if the checker x_i^n is followed by a contact x_{i+1}^n in the cycle then we add the simple directed edges $x_{n(i)}^u \to x_{i+1}^{nr}$ and $x_{i+1}^{nl} \to x_{i+2}^n$. Now by traversing the edges we have just added, the forced matching edges in the direction $x_i^n \to_F x_{n(i)}^u$ and the forced contact edges for the negated part in the direction $x_i^{nr} \to_F x_i^{nl}$, we obtain a cycle that covers all checkers and all the contacts of the negated part, that is, a cycle of direction opposite to C_u^p.

What is left is to encode the equations of size three. Again, we have a central vertex s that is connected to gadgets simulating equations with three variables. For each equation, we create the gadget displayed in Figure 1 (b), which is a variant of the gadget used in [19]. Let $x \oplus y \oplus z = 1$ be the j-th equation with three variables in I_2. This equation is simulated by G_j^{3A}. The vertices used are the contact vertices in $C_j = \{\gamma^\alpha \mid \gamma \in \{x, y, z\}, \alpha \in \{r, l\}\}$, which we have already introduced, as well as the vertices in $H_j = \{s_j, t_j, e_j^i \mid i \in [3]\}$. For notational simplicity, we define $V_j^{3A} = C_j \cup H_j$. All directed non-forced edges are simple. The vertices s_j and t_j are connected to s by forced edges with $w((s, s_j)_F) = w((t_j, s)_F) = \lambda$, where $\lambda > 0$ is a small fixed constant.

Assignment to Tour: We need to construct a tour in G_A given an assignment to the variables of I_2 and prove the following lemma. The proof appears in the fulle version of this paper.

Lemma 3. *Given an instance I_2 of the Hybrid problem with ν bi-wheels and an assignment that leaves k equations in I_2 unsatisfied, then, there exists a tour in G_A with cost at most $37m + 5\nu + 2m\lambda + 2\nu\lambda + k$.*

Tour to Assignment: For the other direction of the reduction we need the following lemma.

Lemma 4. *If there is a tour with cost $37 \cdot m + k + 2\lambda \cdot m$, then, there is an assignment that leaves at most k equations unsatisfied.*

The proof of Lemma 4 and of Theorem 5 are given in the full version of this paper.

7 Concluding Remarks

In this paper, we proved that it is hard to approximate the ATSP and the TSP within any constant factor less than 75/74 and 123/122, respectively. The proof method required essentially new ideas and constructions from the ones used before in that context. Since the best known upper bound on the approximability is $O(\log n/\log\log n)$ for ATSP and 3/2 for TSP, there is certainly room for improvements. Especially, in the asymmetric version of the TSP, there is a large gap between the approximation lower and upper bound, and it remains a major open problem on the existence of an efficient constant factor approximation algorithm for that problem. Furthermore, it would be nice to investigate if some of the ideas of this paper, and in particular the bi-wheel amplifiers, can be used to offer improved hardness results for other optimization problems, such as the Steiner Tree problem.

References

1. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof Verification and the Hardness of Approximation Problems. J. ACM 45, 501–555 (1998)
2. Asadpour, A., Goemans, M., Madry, A., Oveis Gharan, S., Saberi, A.: An $O(\log n/\log\log n)$-Approximation Algorithm for the Asymmetric Traveling Salesman Problem. In: Proc. 21st SODA 2010, pp. 379–389 (2010)
3. Berman, P., Karpinski, M.: On Some Tighter Inapproximability Results. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 200–209. Springer, Heidelberg (1999)
4. Berman, P., Karpinski, M.: Efficient Amplifiers and Bounded Degree Optimization, ECCC TR01-053 (2001)
5. Berman, P., Karpinski, M.: Improved Approximation Lower Bounds on Small Occurrence Optimization, ECCC TR03-008 (2003)

6. Berman, P., Karpinski, M.: 8/7-approximation algorithm for $(1, 2)$-TSP. In: Proc. 17th SODA 2006, pp. 641–648 (2006)
7. Bläser, M.: A 3/4-Approximation Algorithm for Maximum ATSP with Weights Zero and One. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 61–71. Springer, Heidelberg (2004)
8. Böckenhauer, H.-J., Seibert, S.: Improved Lower Bounds on the Approximability of the Traveling Salesman Problem. Theor. Inform. Appl. 34, 213–255 (2000)
9. Christofides, N.: Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, Technical Report CS-93-13, Carnegie Mellon University, Pittsburgh (1976)
10. Engebretsen, L.: An Explicit Lower Bound for TSP with Distances One and Two. Algorithmica 35, 301–318 (2003)
11. Engebretsen, L., Karpinski, M.: TSP with Bounded Metrics. J. Comput. Syst. Sci. 72, 509–546 (2006)
12. Håstad, J.: Some Optimal Inapproximability Results. J. ACM 48, 798–859 (2001)
13. Karpinski, M., Schmied, R.: On Approximation Lower Bounds for TSP with Bounded Metrics, CoRR arXiv: abs/1201.5821 (2012)
14. Karpinski, M., Schmied, R.: On Improved Inapproximability Results for the Shortest Superstring and Related Problems. In: Proc. 19th CATS 2013. CRPIT, vol. 141, pp. 27–36 (2013)
15. Lampis, M.: Improved Inapproximability for TSP. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX 2012 and RANDOM 2012. LNCS, vol. 7408, pp. 243–253. Springer, Heidelberg (2012)
16. Mömke, T., Svensson, O.: Approximating Graphic TSP by Matchings. In: Proc. IEEE 52nd FOCS 2011, pp. 560–569 (2011)
17. Mucha, M.: 13/9-Approximation for Graphic TSP. In: Proc. STACS 2012, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. LIPIcs, vol. 14, pp. 30–41 (2012)
18. Oveis Gharan, S., Saberi, A., Singh, M.: A Randomized Rounding Approach to the Traveling Salesman Problem. In: Proc. IEEE 52nd FOCS 2011, pp. 550–559 (2011)
19. Papadimitriou, C., Vempala, S.: On the Approximability of the Traveling Salesman Problem. In: Proc. 32nd ACM STOC 2000, pp. 126–133 (2000); see also a corrected version in Combinatorica 26, 101–120 (2006)
20. Papadimitriou, C., Yannakakis, M.: The Traveling Salesman Problem with Distances One and Two. Math. Oper. Res. 18, 1–11 (1993)
21. Sebö, A., Vygen, J.: Shorter Tours by Nicer Ears, CoRR arXiv: abs/1201.1870 (2012); to appear in Combinatorica

Smoothed Analysis of the 2-Opt Heuristic for the TSP: Polynomial Bounds for Gaussian Noise

Bodo Manthey and Rianne Veenstra

University of Twente, Department of Applied Mathematics, Enschede, The Netherlands
`b.manthey@utwente.nl`, `h.j.veenstra@student.utwente.nl`

Abstract. The 2-opt heuristic is a very simple local search heuristic for the traveling salesman problem. While it usually converges quickly in practice, its running-time can be exponential in the worst case.

In order to explain the performance of 2-opt, Englert, Röglin, and Vöcking (*Algorithmica*, to appear) provided a smoothed analysis in the so-called one-step model on d-dimensional Euclidean instances. However, translating their results to the classical model of smoothed analysis, where points are perturbed by Gaussian distributions with standard deviation σ, yields a bound that is only polynomial in n and $1/\sigma^d$.

We prove bounds that are polynomial in n and $1/\sigma$ for the smoothed running-time with Gaussian perturbations. In particular our analysis for Euclidean distances is much simpler than the existing smoothed analysis.

1 2-Opt and Smoothed Analysis

The traveling salesman problem (TSP) is one of the classical combinatorial optimization problems. Euclidean TSP is the following variant: given points $X \subseteq [0,1]^d$, find the shortest Hamiltonian cycle that visits all points in X (also called a *tour*). Even this restricted variant is NP-hard for $d \geq 2$ [16]. We consider Euclidean TSP with Manhattan and Euclidean distances as well as squared Euclidean distances to measure the distances between points. For the former two, there exist polynomial-time approximation schemes (PTAS) [1, 14]. The latter, which has applications in power assignment problems for wireless networks [8], admits a PTAS for $d = 2$ and is APX-hard for $d \geq 3$ [15].

As it is unlikely that there are efficient algorithms for solving Euclidean TSP optimally, heuristics have been developed in order to find near-optimal solutions quickly. One very simple and popular heuristic is 2-opt: starting from an initial tour, we iteratively replace two edges by two other edges to obtain a shorter tour until we have found a local optimum. Experiments indicate that 2-opt converges to near-optimal solutions quite quickly [9, 10], but its worst-case performance is bad: the worst-case running-time is exponential even for $d = 2$ [7] and the approximation ratio can be $\Omega(\log n/\log\log n)$ for Euclidean instances [5].

An alternative to worst-case analysis is average-case analysis, where the expected performance with respect to some probability distribution is measured. The average-case running-time for Euclidean instances and the average-case approximation ratio for non-metric instances of 2-opt were analyzed [4–6, 11]. However, while worst-case analysis is often too pessimistic because it is dominated by artificial instances that are rarely encountered in practice, average-case analysis is dominated by random instances, which have often very special properties with high probability that they do not share with typical instances.

In order to overcome the drawbacks of both worst-case and average-case analysis and to explain the performance of the simplex method, Spielman and Teng invented smoothed analysis [17]: an adversary specifies an instance, and then this instance is slightly randomly perturbed. The smoothed performance is the expected performance, where the expected value is taken over the random perturbation. The underlying assumption is that real-world instances are often subjected to a small amount of random noise, which can, e.g., come from measurement or rounding errors. Smoothed analysis often allows more realistic conclusions about the performance than worst-case or average-case analysis. Since its invention, it has been applied successfully to explain the performance of a variety of algorithms [12,18].

Englert, Röglin, and Vöcking [7] provided a smoothed analysis of 2-opt in order to explain its performance. They used the *one-step model*: an adversary specifies n density functions $f_1, \ldots, f_n : [0,1]^d \to [0, \phi]$. Then the n points $x_1, \ldots, x_n$ are drawn independently according to the densities $f_1, \ldots, f_n$, respectively. Here, ϕ is the perturbation parameter. If $\phi = 1$, then the only possibility is the uniform distribution on $[0,1]^d$, and we obtain an average-case analysis. The larger ϕ, the more powerful the adversary. Englert et al. [7] proved that the expected running-time of 2-opt is $O(n^4 \phi \log n)$ and $O(n^{4+\frac{1}{3}} \phi^{\frac{8}{3}} \log^2(n\phi))$ for Manhattan and Euclidean distances, respectively. These bounds can be improved slightly by choosing the initial tour with an insertion heuristic. However, if we transfer these bounds to the classical model of points perturbed by Gaussian distributions of standard deviation σ, we obtain bounds that are polynomial in n and $1/\sigma^d$. This is because the maximum density of a d-dimensional Gaussian with standard deviation σ is $\Theta(\sigma^{-d})$. While this is polynomial for any fixed d, it is unsatisfactory that the degree of the polynomial depends on d.

Our Contribution. We provide a smoothed analysis of the running-time of 2-opt in the classical model, where points in $[0,1]^d$ are perturbed by independent Gaussian distributions of standard deviation σ. The bounds that we prove for Gaussian perturbations are polynomial in n and $1/\sigma$, and the degree of the polynomial is independent of d. As distance measures, we consider Manhattan (Section 3), Euclidean (Section 5), and squared Euclidean distances (Section 4).

The analysis for Manhattan distances is a straightforward adaptation of the existing analysis by Englert et al. However, while the degree of the polynomial in n is independent of d in our bound, we still have a factor in the bound that is exponential in d.

Our analysis for Euclidean distances is considerably simpler than the one by Englert et al., which is rather technical and takes more than 20 pages [7].

The analysis for squared Euclidean distances is, to our knowledge, not preceded by a smoothed analysis in the one-step model. Because of the nice properties of squared Euclidean distances and Gaussian perturbations, this smoothed analysis is relatively compact and elegant: the only concept needed for

Theorem 4.3 is pairs of linked 2-changes (Section 2.1), and we can even get rid of this at the price of a slightly worse bound (Remark 4.4). This might be of independent interest, as smoothed analysis of local search heuristics is often rather technical [2,3,7,13].

We did not try to optimize our bounds, but rather tried to keep the analysis simple. We believe that much stronger bounds hold for Euclidean and squared Euclidean distances (see also Section 6).

2 Notation, Preliminaries and Outline

Throughout the rest of this paper, X denotes a set of n points in $\mathbb{R}^d$, where each point is drawn according to an independent d-dimensional Gaussian distribution with mean in $[0,1]^d$ and standard deviation σ. The dimension d is considered to be constant. We discuss the dependence of our bounds on d in Section 6.

We assume that $\sigma \leq 1$. This is justified by two reasons. First, small σ are actually the interesting case, i.e., when the order of magnitude of the perturbation is relatively small. Second, the smoothed number of iterations that 2-opt needs is a monotonically decreasing function of σ: if we have $\sigma > 1$, then this is equivalent to adversarial instances in $[0,1/\sigma]^d$ that are perturbed with standard deviation 1. This in turn is dominated by adversarial instances in $[0,1]^d$ that are perturbed with standard deviation 1, as $[0,1/\sigma]^d \subseteq [0,1]^d$. Thus, any bound for $\sigma = 1$ holds also for larger σ. Sometimes we even assume $\sigma = O(1/\sqrt{n \log n})$ to simplify the analysis.

2.1 2-Opt State Graph and Linked 2-Changes

Given a tour H that visits all points in X, a *2-change* replaces two edges $\{x_1, x_2\}$ and $\{x_3, x_4\}$ of H by two new edges $\{x_1, x_3\}$ and $\{x_2, x_4\}$, provided that this yields again a tour (this is the case if x_1, x_2, x_3, x_4 appear in this order in the tour) and that this decreases the length of the tour, i.e., $d(x_1, x_2) + d(x_3, x_4) - d(x_1, x_3) - d(x_2, x_4) > 0$, where $d(a, b) = \|a - b\|_2$ (Euclidean distances), $d(a, b) = \|a - b\|_1$ (Manhattan distances), or $d(a, b) = \|a - b\|_2^2$ (squared Euclidean distances). The 2-opt heuristic iteratively improves an initial tour by applying 2-changes until it reaches a local optimum.

The number of iterations that 2-opt needs depends of course heavily on the initial tour and on which 2-change is chosen in each iteration. We do not make any assumptions about the initial tour and about which 2-change is chosen. Following Englert et al. [7], we consider the *2-opt state graph*: we have a node for every tour and a directed edge from tour H to tour H' if H' can be obtained by one 2-change. The 2-opt state graph is a directed acyclic graph, and the length of the longest path in the 2-opt state graph is an upper bound for the number of iterations that 2-opt needs.

In order to improve the bounds (for Manhattan distances) or to allow bounds on the expected number of iterations in the first place (for Euclidean and squared Euclidean distances), we also consider *pairs of linked 2-changes* [7]. Two 2-changes form a pair of linked 2-changes if there is one edge added in one 2-change and removed in the other 2-change. Formally, one 2-change replaces $\{x_1, x_2\}$ and $\{x_3, x_4\}$ by $\{x_1, x_3\}$ and $\{x_2, x_4\}$ and the other 2-change replaces $\{x_1, x_3\}$ and $\{x_5, x_6\}$ by $\{x_1, x_5\}$ and $\{x_2, x_6\}$. It can be that $\{x_2, x_4\}$ and $\{x_5, x_6\}$ intersect. Englert et al. [7] called a pair of linked 2-changes a *type i pair* if $|\{x_2, x_4\} \cap \{x_5, x_6\}| = i$. As type 2 pairs, which involve in fact only four nodes, are difficult to analyze because of dependencies, we ignore them. Fortunately, the following lemma states that we will find enough disjoint pairs of linked 2-changes of type 0 and 1 in any sufficiently long sequence of 2-changes.

Lemma 2.1 (Englert et al. [7]). *Every sequence of t consecutive 2-changes contains at least $t/6 - 7n(n-1)/24$ disjoint pairs of linked 2-changes of type 0 or type 1.*

2.2 Technical Lemmas

In order to get an upper bound for the length of the initial tour, we need an upper bound for the diameter of the point set X. Such an upper bound is also necessary for the analysis of 2-changes with Euclidean distances (Section 5). We choose $D_{\max}$ such that $X \subseteq [-D_{\max}, D_{\max}]^d$ with a probability of at least $1 - 1/n!$. For fixed d and $\sigma \leq 1$, we can choose $D_{\max} = \Theta(1 + \sigma\sqrt{n \log n})$ according to the following lemma. For $\sigma = O(1/\sqrt{n \log n})$, we have $D_{\max} = \Theta(1)$.

Lemma 2.2. *Let $c > 0$ be a sufficiently large constant, and let $D_{\max} = c \cdot (\sigma\sqrt{n \log n} + 1)$. Then $\mathbb{P}(X \not\subseteq [-D_{\max}, D_{\max}]^d) \leq 1/n!$.*

We need the following simple fact a few times.

Lemma 2.3 (Arthur et al. [2, Fact 2.1]). *Let $p \in [0, 1]$ be a probability, and let a, c, b, d, and e be non-negative real numbers with $c \geq 1$ and $e \geq d$. If $p \leq a + c \cdot b^e$, then $p \leq a + c \cdot b^d$.*

For $x, y \in \mathbb{R}^d$ with $x \neq y$, let $L(x, y) = \{\xi \cdot (y - x) + x \mid \xi \in \mathbb{R}\}$ denote the straight line through x and y.

Lemma 2.4. *Let $a, b \in \mathbb{R}^d$ be arbitrary with $a \neq b$. Let $c \in \mathbb{R}^d$ be drawn according to a d-dimensional Gaussian distribution with standard deviation σ. Then the probability that c is ε-close to $L(a, b)$, i.e., $\min_{c^\star \in L(a,b)} \|c - c^\star\|_2 \leq \varepsilon$, is bounded from above by $(\varepsilon/\sigma)^{d-1}$.*

Let $\delta_{\text{close}} = \min_{a,b \in X, a \neq b} \|a - b\|_2$ be the minimum distance of points in X.

Lemma 2.5. *For any $\varepsilon > 0$, we have $\mathbb{P}(\delta_{\text{close}} \leq \varepsilon) \leq n^2 \cdot (\varepsilon/\sigma)^d$.*

We need the following lemma in Section 5.

Lemma 2.6. *Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function whose derivative is bounded from above by B, let c be distributed according to Gaussian distribution with standard deviation σ. Let I be an interval of size ε, and let $f(I) = \{f(x) \mid x \in I\}$ be the image of I. Then $\mathbb{P}(c \in f(I)) = O(B\varepsilon/\sigma)$.*

2.3 Outline

The main idea in the proofs by Englert et al. [7] and in our proofs is to bound the minimal improvement of any 2-change or, for pairs of linked 2-changes, the minimal improvement of any pair of linked 2-changes. We denote the smallest improvement of any linked 2-change by $\Delta_{\min}$ and the smallest improvement of any pair of linked 2-changes by $\Delta_{\min}^{\text{link}}$. It will be clear from the context which distance measure is used for $\Delta_{\min}$ and $\Delta_{\min}^{\text{link}}$. Suppose that the initial tour has a length of at most L, then 2-opt cannot run for more than $L/\Delta_{\min}$ iterations and not for more than $\Theta(L/\Delta_{\min}^{\text{link}})$ iterations. The following lemma formalizes this.

Lemma 2.7. *Suppose that, with a probability of at least* $1 - 1/n!$, *any tour has a length of at most* L. *Let* $\gamma > 1$. *Then*

1. *If* $\mathbb{P}(\Delta_{\min} \leq \varepsilon) = O(P\varepsilon)$, *then the expected length of the longest path in the 2-opt state graph is bounded from above by* $O(PLn \log n)$.
2. *If* $\mathbb{P}(\Delta_{\min} \leq \varepsilon) = O(P\varepsilon^{\gamma})$, *then the expected length of the longest path in the 2-opt state graph is bounded from above by* $O(P^{1/\gamma}L)$.
3. *The same bounds hold if we replace* $\Delta_{\min}$ *by* $\Delta_{\min}^{\text{link}}$, *provided that* $PL = \Omega(n^2)$ *for Case 1 and* $P^{1/\gamma}L = \Omega(n^2)$ *for Case 2.*

For Euclidean and squared Euclidean distances, it turns out to be useful to study $\Delta_{a,b}(c) = d(c,a) - d(c,b)$ for points $a, b, c \in X$. By abusing notation, we sometimes write $\Delta_{i,j}(k)$ instead of $\Delta_{x_i,x_j}(x_k)$ for short. A 2-change that replaces $\{x_1, x_2\}$ and $\{x_3, x_4\}$ by $\{x_1, x_3\}$ and $\{x_2, x_4\}$ improves the tour length by $\Delta_{1,4}(2) - \Delta_{1,4}(3) = \Delta_{2,3}(1) - \Delta_{2,3}(4)$.

3 Manhattan Distances

The analysis for Manhattan distances is a straightforward adaptation of the analysis in the one-step model. We obtain a bound of $O(n^4 D_{\max}/\sigma)$. The term $D_{\max}$ in the bound comes from the bound of the initial tour.

Lemma 3.1. $\mathbb{P}(\Delta_{\min}^{\text{link}} \leq \varepsilon) = O(n^6\varepsilon^2/\sigma^2)$.

Theorem 3.2. *The expected length of the longest path in the 2-opt state graph corresponding to d-dimensional instances with Manhattan distances is at most* $O(n^4 D_{\max}/\sigma)$.

4 Squared Euclidean Distances

In this section, we have $\Delta_{a,b}(c) = \|c - a\|_2^2 - \|c - b\|_2^2$ for $a, b, c \in \mathbb{R}^d$.

Lemma 4.1. *Let* $a, b \in \mathbb{R}^d$, $a \neq b$, *and let* c *be drawn according to a Gaussian distribution with standard deviation* σ. *Let* $I \subseteq \mathbb{R}$ *be an interval of length* ε. *Then* $\mathbb{P}\big(\Delta_{a,b}(c) \in I\big) \leq \frac{\varepsilon}{4\sigma \cdot \|a-b\|_2}$.

Proof. Since Gaussian distributions are rotationally symmetric, we can assume without loss of generality that $a = (0, \ldots, 0)$ and $b = (\delta, 0, \ldots, 0)$. We have $\delta = \|a - b\|_2$. Let $c = (c_1, \ldots, c_d)$. Then $\varDelta_{a,b}(c) = c_1^2 - (c_1 - \delta)^2 = 2c_1\delta + \delta^2$. Thus, $\varDelta_{a,b}(c)$ falls into I if and only if c_1 falls into an interval of length $\frac{\varepsilon}{2\delta}$. Since c_1 is a 1-dimensional Gaussian random variable with a standard deviation of σ, the probability for this is bounded from above by $\frac{\varepsilon}{4\delta\sigma}$. □

We analyze $\varDelta_{\min}^{\text{link}}$ since it seems to be difficult to obtain bounds for the expected value using $\varDelta_{\min}$.

Lemma 4.2. *For $d \geq 2$, we have $\mathbb{P}(\varDelta_{\min}^{\text{link}} \leq \varepsilon) = O(n^6 \varepsilon \sigma^{-2})$.*

Proof. Consider a pair of linked 2-changes where $\{x_1, x_2\}$ and $\{x_3, x_4\}$ are replaced by $\{x_1, x_3\}$ and $\{x_2, x_4\}$ and then $\{x_1, x_3\}$ and $\{x_5, x_6\}$ by $\{x_1, x_5\}$ and $\{x_3, x_6\}$. We assume that $x_2 \notin \{x_5, x_6\}$ and $x_5 \notin \{x_2, x_4\}$. The other cases are identical.

If the pair yields an improvement of at most ε then $\varDelta_{1,3}(2)$ falls into some interval of length at most ε and $\varDelta_{1,6}(5)$ falls into some interval of length at most ε. We have $\|x_1 - x_3\|_2 \leq \sqrt{\varepsilon}$ or $\|x_1 - x_6\|_2 \leq \sqrt{\varepsilon}$ only if $\delta_{\text{close}} \leq \sqrt{\varepsilon}$, which happens with a probability of at most $n^2(\sqrt{\varepsilon}/\sigma)^d \leq n^2 \varepsilon \sigma^{-2}$ by Lemmas 2.5 and 2.3 since $d \geq 2$. From now on, we assume that $\|x_1 - x_3\|_2, \|x_1 - x_6\|_2 \geq \sqrt{\varepsilon}$. By independence of x_2 and x_5 and Lemma 4.1, the probability that both $\varDelta_{1,3}(2)$ and $\varDelta_{1,6}(5)$ fall into their "bad" interval of length ε is thus bounded from above by $\left(\frac{\sqrt{\varepsilon}}{4\sigma}\right)^2 = O(\varepsilon\sigma^{-2})$.

The lemma follows now by a union bound over all $O(n^6)$ pairs of linked 2-changes and the fact that we do not have to apply the union bound to the probability that δ_{close} is small. □

Theorem 4.3. *For $d \geq 2$, the expected length of the longest path in the 2-opt state graph corresponding to d-dimensional instances with squared Euclidean distances is at most $O(n^8 \log n \cdot D_{\max}^2/\sigma^2)$.*

Proof. The theorem follows by using Lemma 2.7 with Lemma 4.2 and the observation that the initial tour has a length of at most $O(D_{\max}^2 n)$ with a probability of at least $1 - 1/n!$ by Lemma 2.2. □

Remark 4.4. The proof of Theorem 4.3 can be simplified by getting rid of the pairs of linked 2-changes (Lemma 2.1) and slightly worsening the bound to $O(n^{10} \log n \cdot D_{\max}^2/\sigma^2)$: we observe that two consecutive 2-changes involve between five and eight nodes as they cannot involve the same four points. Thus, there is always one node that is only involved in the first of the two and one node that is only involved in the second of the two. This is sufficient but worsens the bound of Lemma 4.2 as we have to take a union bound over $O(n^8)$ choices for the two 2-changes instead of $O(n^6)$ choices for pairs of linked 2-changes.

5 Euclidean Distances

In this section, we have $\varDelta_{a,b}(c) = \|c - a\|_2 - \|c - b\|_2$ for $a, b, c \in \mathbb{R}^d$. Analyzing $\|c - a\|_2 - \|c - b\|_2$ turns out to be more difficult than analyzing $\|c - a\|_2^2 - \|c - b\|_2^2$

in the previous section. In particular the case when $\|c-a\|_2 - \|c-b\|_2$ is close to the maximal possible value of $\|a-b\|_2$ requires special attention.

5.1 Difference of Euclidean Distances

As for squared Euclidean distances, we analyze the probability that a pair of linked 2-changes yields a small improvement. Assume that a, b, and c are already drawn. Then the 2-change that replaces $\{z,a\}$ and $\{b,c\}$ by $\{z,b\}$ and $\{a,c\}$ yields an improvement of at most ε only if $\eta = \|z-a\|_2 - \|z-b\|_2 = \varDelta_{a,b}(z)$ falls in a particular interval of length ε. For this analysis, it does not matter which of the four points involved in the 2-change is chosen as z.

We observe that η is essentially 2-dimensional: it depends only on the distance of z from $L(a,b)$ (this is x in the following lemma) and on the position of the projection z to $L(a,b)$ (this is y in the following lemma). Furthermore, it depends on the distance $\|a-b\|_2$ between a and b (this is δ in the following lemma). The following lemma makes the connection between x and y explicit for a given η.

Lemma 5.1. *Let $z = (x,y) \in \mathbb{R}^2$, $x \geq 0$, $y \geq 0$. Let $a = (0,-\delta/2)$ and $b = (0,\delta/2)$ be two points at a distance of δ. Let $\eta = \|z-a\|_2 - \|z-b\|_2$. Then we have*

$$y^2 = \frac{\eta^2\delta^2 + 4\eta^2x^2 - \eta^4}{4\delta^2 - 4\eta^2} = \frac{\eta^2}{4} + \frac{\eta^2x^2}{\delta^2-\eta^2} \tag{1}$$

for $0 \leq \eta < \delta$ and

$$x^2 = \frac{y^2\cdot\left(4\delta^2 - 4\eta^2\right) + \eta^4 - \eta^2\delta^2}{4\eta^2} = \frac{y^2\cdot\left(\delta^2-\eta^2\right)}{\eta^2} - \frac{\delta^2-\eta^2}{4}. \tag{2}$$

for $\delta \geq \eta > 0$. Furthermore, $\eta > \delta$ is impossible.

In order to apply Lemma 2.6, we need the following upper bound on the derivative of y with respect to η, given that x is fixed.

Lemma 5.2. *For $x, y \geq 0$, let $y = \sqrt{\frac{\eta^2}{4} + \frac{\eta^2x^2}{\delta^2-\eta^2}}$. Assume that $0 \leq \eta \leq \delta - \kappa$ and $\kappa > 0$. Then the derivative of y with respect to η is bounded by $O\left(\frac{\delta^2+x^2}{\kappa^2}\right)$.*

If δ and x are bounded by $O(D_{\max})$, then the derivative of y with respect to η is bounded by $O(D_{\max}^2/\kappa^2)$.

We stress that Lemma 5.2 provides a rather bad upper bound on the derivative: We use an upper bound of $D_{\max}$ for x in the numerator, while $x \approx D_{\max}$ would lead to a much larger denominator and, thus, to a better bound. However, we try to keep the analysis simple, and it seems difficult to get a better compact upper bound for the derivative without case distinctions.

Using Lemmas 5.2 and 2.6, we can bound the probability that $\varDelta_{a,b}(z)$ assumes a value in an interval of size ε.

Lemma 5.3. *Let $a, b \in [-D_{\max}, D_{\max}]^d$ be arbitrary, $a \neq b$, and let z be drawn according to a Gaussian distribution with standard deviation σ. Let $\delta = \|a-b\|_2 = O(D_{\max})$. Let I be an interval of length ε with $I \subseteq [0, \delta-\kappa]$. Then*

$$\mathbb{P}\left(\varDelta_{a,b}(z) \in I \text{ and } z \in [-D_{\max}, D_{\max}]^d\right) = O(\varepsilon D_{\max}^2\kappa^{-2}\sigma^{-1}).$$

5.2 Bad Events

Lemma 5.2 and, thus, Lemma 5.3 become quite weak if κ is small. This is the case if $\Delta_{a,b}(z)$ is close to its maximal possible value of $\|a-b\|_2$. In this case, z must be very close to $L(a,b)$. The following lemma states that this is unlikely.

Lemma 5.4. *For $d \geq 2$ and $0 < \alpha < \beta < 1$, let $E_{\varepsilon,\alpha,\beta}$ be the event that at least one of the following bad events occur:*

1. *$X \nsubseteq [-D_{\max}, D_{\max}]^d$.*
2. *$\delta_{\text{close}} \leq \varepsilon^\alpha$.*
3. *There exist four different points $a, b, c, c' \in X$ with $|\Delta_{a,b}(c)| \geq \|a-b\|_2 - \varepsilon^\beta$ and $|\Delta_{a,b}(c')| \geq \|a-b\|_2 - 2\varepsilon^\beta$.*

Then, for all $\varepsilon \leq \varepsilon_0$ for some ε_0 that depends on α and β, we have

$$\mathbb{P}(E_{\varepsilon,\alpha,\beta}) \leq \frac{1}{n!} + n^2 \cdot \left(\frac{\varepsilon^\alpha}{\sigma}\right)^d + n^4 \cdot \left(\frac{8\varepsilon^{\beta-\alpha} D_{\max}^2}{\sigma^2}\right)^{d-1}$$

Proof (sketch). The three terms of the bound correspond to the three parts of the bad events. The first two are immediate consequences of Lemmas 2.2 and 2.5.

For the last term and Item 3, we observe that, because $X \subseteq [-D_{\max}, D_{\max}]^d$, the event $\Delta_{a,b}(c) \geq \|a-b\|_2 - \varepsilon^\beta$ can only occur if c is within a distance of at most $2\sqrt{\varepsilon^{\alpha-\beta}} D_{\max}$ of $L(a,b)$. The probability that this happens can be bounded using Lemma 2.4. In the same way, the probability of the event $\Delta_{a,b}(c') \geq \|a-b\|_2 - 2\varepsilon^\beta$ can be bounded from above. □

5.3 Smallest Improvement of a Pair of Linked 2-Changes

In this section, we analyze the probability that there exists a pair of linked 2-changes that yields an improvement of at most ε. Simple 2-changes do not seem sufficient to yield a bound on the expected number of iterations.

Lemma 5.5. *Fix α and β, and let $\varepsilon > 0$ be sufficiently small as in Lemma 5.4. Then*

$$\mathbb{P}\big(\Delta_{\min}^{\text{link}} < \varepsilon \text{ and not } E_{\varepsilon,\alpha,\beta}\big) = O\left(n^6 \varepsilon^{2-4\beta} D_{\max}^4 \sigma^{-2}\right).$$

Proof. We analyze a fixed pair of linked 2-changes as described in Section 2.1. Then the lemma follows by a union bound over the $O(n^6)$ possible pairs. We assume that $\delta_{\text{close}} \geq \varepsilon^\alpha$. Otherwise, we would have event $E_{\varepsilon,\alpha,\beta}$ (Lemma 5.4, Item 2).

Suppose that $|\Delta_{1,4}(3)| \geq \|x_1 - x_3\| - \varepsilon^\beta$. Then, because we do not have $E_{\varepsilon,\alpha,\beta}$ (Lemma 5.4, Item 3), we have $|\Delta_{1,4}(2)| \leq \|x_1 - x_3\| - 2\varepsilon^\beta$. The improvement of the first 2-change of the linked pair is $|\Delta_{1,4}(2) - \Delta_{1,4}(3)| \geq \varepsilon^\beta \geq \varepsilon$ or it is not a 2-change as there is no improvement. In the same way, if $|\Delta_{1,4}(2)| \geq \|x_1 - x_3\| - \varepsilon^\beta$ or $\Delta_{1,6}(3) \geq \|x_1 - x_6\| - \varepsilon^\beta$ or $\Delta_{1,6}(5) \geq \|x_1 - x_6\| - \varepsilon^\beta$, at least one of the two 2-changes yields an improvement of at least $\varepsilon^\beta \geq \varepsilon$. Thus, we can ignore these cases from now on and apply Lemma 5.3 with $\kappa = \varepsilon^\beta$.

We first draw all but two of the five or six points (depending on which type of linked pair we have) such that one of the two remaining points (x_i with $i \in \{2,4\}$) is only involved in the first 2-change and the other point (x_j with $j \in \{5,6\}$) is only involved in the second 2-change. We only consider the case $i = 2$ and $j = 5$, the other cases are identical.

The first 2-change yields an improvement of at most ε only if $\Delta_{1,4}(2)$ falls into an interval of size at most ε. According to Lemma 5.3, the probability for this is at most $O(\varepsilon^{1-2\beta} D_{\max}^2 \sigma^{-1})$, as we have already ruled out the case that $X \not\subseteq [-D_{\max}, D_{\max}]^d$. Analogously, the probability that $\Delta_{1,6}(5)$ falls into an interval of length at most ε is at most $O(\varepsilon^{1-2\beta} D_{\max}^2 \sigma^{-1})$, and this is necessary for the second 2-change to yield an improvement of at most ε. By independence of x_2 and x_5, the probability that none of the two 2-changes yields an improvement of at least ε and that we do not have event $E_{\varepsilon,\alpha,\beta}$ is bounded from above by $O(\varepsilon^{2-4\beta} D_{\max}^4 \sigma^{-2})$. □

The following lemma is an immediate consequence of Lemmas 5.4 and 5.5.

Lemma 5.6. *For any* $0 < \alpha < \beta < 1$, *we have*

$$\mathbb{P}\big(\Delta_{\min}^{\text{link}} \leq \varepsilon \text{ and } X \subseteq [-D_{\max}, D_{\max}]^d\big)$$
$$= O\left(n^6 \cdot \frac{\varepsilon^{2-4\beta} D_{\max}^4}{\sigma^2} + n^2 \cdot \left(\frac{\varepsilon^{\alpha}}{\sigma}\right)^d + n^4 \cdot \left(\frac{\varepsilon^{\beta-\alpha} D_{\max}^2}{\sigma^2}\right)^{d-1}\right).$$

Now we choose $\beta = 0.247$ and $\alpha = 0.12$. Then, for $d \geq 9$, this yields $2 - 4\beta > 1.01$, $\alpha d > 1.08$, and $(\beta - \alpha) \cdot (d-1) > 1.01$. Using Lemma 2.3, this allows us to remove the d from the exponent, and we obtain the following simplified version of Lemma 5.6. We assume that $\sigma = O(1/\sqrt{n \log n})$ for simplicity. Thus, $D_{\max} = O(1)$.

Lemma 5.7. *For* $d \geq 9$ *and* $\sigma = O(1/\sqrt{n \log n})$, *we have* $\mathbb{P}\big(\Delta_{\min}^{\text{link}} \leq \varepsilon \text{ and } X \subseteq [-D_{\max}, D_{\max}]^d\big) = O(\varepsilon^{1.01} n^4 \sigma^{-16})$.

Using this lemma, we can prove the main result of this section.

Theorem 5.8. *For* $d \geq 9$ *and* $\sigma = O(1/\sqrt{n \log n})$, *the expected length of the longest path in the 2-opt state graph corresponding to d-dimensional instances with Euclidean distances is at most* $O(n^5/\sigma^{16})$.

6 Concluding Remarks

Improving the bounds. Our smoothed analysis for Euclidean instances works only for $d \geq 9$ and the dependence of the bound on σ is bad. With the same analysis, we can get a better bound – in particular with respect to σ – for larger values of d by adjusting Lemma 5.7. While our goal was to keep the analysis simple, we believe that a much better bound holds, also for smaller d, by exploiting techniques of Englert et al. [7] for Euclidean distances.

Similarly, we can obtain an improved bound for squared Euclidean distances by considering $d \geq 3$ and adapting Lemma 4.2.

Polynomial bound for Euclidean distances for all d. For $d \leq 8$, the bound proved by Englert et al. [7] for Euclidean distances is $O(n^{4+\frac{1}{3}} \log(n/\sigma)\sigma^{-21.4})$. By combining this with our bound, we obtain a smoothed polynomial number of iterations for all d and without d in the exponent.

Initial tour. One reason that we obtain worse bounds is that our upper bound for the length of the initial tour is worse because we do not truncate the Gaussian distributions. This effect is even stronger for Euclidean distances, where the maximum distance between points plays a role also in the analysis of the 2-changes (Lemmas 5.3 and 5.4). Only for $\sigma = O(1/\sqrt{n \log n})$, this effect is negligible, as then $D_{\max} = O(1)$.

In the same way as Englert et al. [7], we can slightly improve the smoothed number of iterations by using an insertion heuristic to choose the initial tour. We save a factor of $n^{1/d}$ for Manhattan and Euclidean distances and a factor of $n^{2/d}$ for squared Euclidean distances. The reason is that there always exist tours of length $O(D_{\max} n^{1-\frac{1}{d}})$ for n points in $[-D_{\max}, D_{\max}]^d$ for Euclidean and Manhattan distances and of length $O(D_{\max}^2 n^{1-\frac{2}{d}})$ for squared Euclidean distances for $d \geq 2$ [19].

Dependence on d. For Manhattan distances, the term hidden in the O depends exponentially on d. For Euclidean distances, the dependence is polynomially on d. For squared Euclidean distances, the term depends only linearly on d.

We conjecture that also for Manhattan distances, a bound that avoids exponential dependence on d can be proved.

Approximation ratio. Using the fact that any local optimum of 2-opt yields a tour of length at most $O(D_{\max} n^{1-\frac{1}{d}})$ [5] and that the optimal tour has a length of $\Omega(n^{1-\frac{1}{d}}\sigma)$ [7], we obtain a smoothed approximation ratio of $O(D_{\max}/\sigma)$. This, however, is worse than the worst-case ratio of $O(\log n)$ [5] as $D_{\max}/\sigma = \Omega(\sqrt{n/\log n})$. The reason for this bound is that the upper bound for the local optimum involves $D_{\max}$.

We conjecture an approximation ratio of $O(1/\sigma)$, which is what we would obtain if plugging $\sigma = \Theta(\phi^{-d})$ into the bound of Englert et al. [7] were allowed.

References

1. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
2. Arthur, D., Manthey, B., Röglin, H.: Smoothed analysis of the k-means method. Journal of the ACM 58(5) (2011)
3. Arthur, D., Vassilvitskii, S.: Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. SIAM Journal on Computing 39(2), 766–782 (2009)
4. Bringmann, K., Engels, C., Manthey, B., Rao, B.V.R.: Random shortest paths: Non-euclidean instances for metric optimization problems. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 219–230. Springer, Heidelberg (2013)

5. Chandra, B., Karloff, H., Tovey, C.: New results on the old k-opt algorithm for the traveling salesman problem. SIAM Journal on Computing 28(6), 1998–2029 (1999)
6. Engels, C., Manthey, B.: Average-case approximation ratio of the 2-opt algorithm for the TSP. Operations Research Letters 37(2), 83–84 (2009)
7. Englert, M., Röglin, H., Vöcking, B.: Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. Algorithmica (to appear)
8. Funke, S., Laue, S., Lotker, Z., Naujoks, R.: Power assignment problems in wireless communication: Covering points by disks, reaching few receivers quickly, and energy-efficient travelling salesman tours. Ad Hoc Networks 9(6), 1028–1035 (2011)
9. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, ch. 8. John Wiley & Sons (1997)
10. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and its Variations, ch. 9. Kluwer Academic Publishers (2002)
11. Kern, W.: A probabilistic analysis of the switching algorithm for the TSP. Mathematical Programming 44(2), 213–219 (1989)
12. Manthey, B., Röglin, H.: Smoothed analysis: Analysis of algorithms beyond worst case. it – Information Technology 53(6), 280–286 (2011)
13. Manthey, B., Röglin, H.: Worst-case and smoothed analysis of k-means clustering with Bregman divergences. Journal of Computational Geometry 4(1), 94–132 (2013)
14. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. SIAM Journal on Computing 28(4), 1298–1309 (1999)
15. van Nijnatten, F., Sitters, R., Woeginger, G.J., Wolff, A., de Berg, M.: The traveling salesman problem under squared Euclidean distances. In: Proc. of the 27th Int. Symp. on Theoretical Aspects of Computer Science (STACS). LIPIcs, vol. 5, pp. 239–250. Schloss Dagstuhl (2010)
16. Papadimitriou, C.H.: The Euclidean traveling salesman problem is NP-complete. Theoretical Computer Science 4(3), 237–244 (1977)
17. Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. Journal of the ACM 51(3), 385–463 (2004)
18. Spielman, D.A., Teng, S.H.: Smoothed analysis: An attempt to explain the behavior of algorithms in practice. Communications of the ACM 52(10), 76–84 (2009)
19. Yukich, J.E.: Probability Theory of Classical Euclidean Optimization Problems. Lecture Notes in Mathematics, vol. 1675. Springer (1998)

Tight Approximation Bounds for Connectivity with a Color-Spanning Set[⋆]

Chenglin Fan[1], Jun Luo[1,2], and Binhai Zhu[3]

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
{cl.fan,jun.luo}@siat.ac.cn
[2] Huawei Noah's Ark Laboratory, Hong Kong
[3] Department of Computer Science, Montana State University, Bozeman, MT 59717, USA
bhz@cs.montana.edu

Abstract. Given a set of points Q in the plane, define the $\frac{r}{2}$-*Disk Graph*, $Q(r)$, as a generalized version of the Unit Disk Graph: the vertices of the graph is Q and there is an edge between two points in Q iff the distance between them is at most r. In this paper, motivated by applications in wireless sensor networks, we study the following geometric problem of color-spanning sets: given n points with m colors in the plane, choosing m points P with distinct colors such that the $\frac{r}{2}$-Disk Graph, $P(r)$, is connected and r is minimized. When at most two points are of the same color c_i (or, equivalently, when a color c_i spans at most two points), we prove that the problem is NP-hard to approximate within a factor $3 - \varepsilon$. And we present a tight factor-3 approximation for this problem. For the more general case when each color spans at most k points, we present a factor-($2k$-1) approximation. Our solutions are based on the applications of the famous Hall's Marriage Theorem on bipartite graphs, which could be useful for other problems.

1 Introduction

In a wireless sensor network (WSN), the typical objective is to use a set of sensors (modelled as unit disks) to cover a region (or a set of objects) completely. However, in many situations this is either impossible or too costly to achieve, like in a battlefield or in a vast rural area. Hence, recently *partial covers* are proposed to cover a region (or a set of objects) with a decent quality guarantee [18, 27]. (It is well-known that in WSNs the communication range is greater than the sensing range and if the former is at least twice the latter then a complete coverage implies a communication connectivity [26].) Certainly, in partial covers the sensors are usually disconnected (within their sensing range), so we need to increase the communication range (radius) to make the whole WSN connected — which certainly takes energy.

In Figure 1, we show a partial cover with three connected components/clusters A, B and C. To save energy, we just need to select three leaders a, b, c respectively so that by increasing the communication range of these leaders they can communicate with each

[⋆] This research is partially funded by the International Science and Technology Cooperation Program of China (2010DFA92720-08-1) and NSF of China under project 11271351.

other and possibly relay some important sensing data from each cluster. In fact, in a more general and slightly different setting, say, in a social network where the members in each group can communicate in different ways, the clusters could even be interleaved and might be inseparable geometrically.

This is often referred to as the color-spanning problem in computational geometry, usually to handle imprecise data. In this model, each imprecise point is modelled as a set T of discrete points which can all be painted by one distinct color c_i (we also say that the color c_i *spans* T). The causes of imprecise data can be various, for example, the uncertain properties of a moving object [5], measurement error, sampling error, network latency [20,21], location privacy protection [3,6,12], etc. Any or a combination of these factors could be leading to imprecision of the data, hence such a new model makes sense for many applications.

In the database area, a similar framework under a different name "uncertain data" has also been used. An imprecise point is called an uncertain object and the different positions with the same color are regarded as the different possible instances of an uncertain object. Pei *et al.* have performed some research that pertains to geometric problems in this framework [4,19,24].

In general, the color-spanning problem is to select exactly one point from each colored point set such that certain properties (e.g. area, distance, perimeter, etc) of some underlying geometric structures (e.g. convex hulls, minimum spanning trees, etc) based on the selected points with different colors are minimized or maximized. We give a brief review for some works in computational geometry below.

In the following review, we assume that there are n points with m colors for the sake of notation consistency. Zhang *et al.* [25] proposed a brute force algorithm to address the minimum diameter color-spanning set problem (MDCS). The running time is $O(n^m)$. Fleischer and Xu [11] showed that the MDCS problem can be solved in polynomial time for the L_1 and L_∞ metrics, while it is NP-hard for all other L_p metrics (even for $p = 2$). They also gave an efficient algorithm to compute a constant factor approximation.

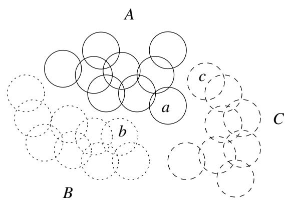


Fig. 1. Three connected components A, B and C for a partial cover. a, b and c can be selected to use the minimum energy to make them connected via communication.

Abellanas *et al.* [1] showed that the Farthest Color Voronoi Diagram (FCVD) is of complexity $\Theta(nm)$ if $m \leq n/2$. Then they proposed algorithms to construct FCVD, the smallest color-spanning circle based on FCVD, the smallest color-spanning rectangle and the narrowest color-spanning strip of arbitrary orientation. In [2], Abellanas *et al.* also proposed an $O(\min\{n(n-m)^2, nm(n-m)\})$ time algorithm for computing the smallest perimeter axis-parallel rectangle enclosing at least one point of each color.

In [9], Das *et al.* proposed an algorithm for identifying the smallest color-spanning corridor in $O(n^2 \log n)$ time and $O(n)$ space and an algorithm for identifying the smallest color-spanning rectangle of arbitrary orientation with an $O(n^3 \log m)$ running time and $O(n)$ space.

Ju *et al.* [16] recently studied several other color-spanning problems. They gave an efficient randomized algorithm to compute a maximum diameter color-spanning set, and they showed it is NP-hard to compute a largest closest pair color-spanning set and a planar minimum color-spanning tree.

Given a set of points Q in the plane, define the $\frac{r}{2}$-*Disk Graph*, $Q(r)$, as a generalized version of the Unit Disk Graph as follows: the vertex set of the graph is Q and there is an edge between two points in Q iff the distance between them is at most r. For a Unit Disk Graph, we have $r = 2$.

In this paper, we study the following color-spanning set problem: The input is a set S of n points with m colors in the plane. We want to choose m points P from S with m distinct colors such that the $\frac{r}{2}$-Disk Graph on P, $P(r)$, is connected and r is minimized. We call this problem the *minimum connected color-spanning set* problem, abbreviated as **MCCS**. When each color spans at most k points, the problem is denoted as $MCCS(k)$.

While this problem is new, it resembles some of the previous research on "Minimum Spanning Tree with Neighborhoods", etc. Interested readers are referred to [10, 23].

We summarize our results as follows.

1. MCCS(2) is NP-hard to approximate within a factor $3 - \varepsilon$, for some $\varepsilon > 0$.
2. For MCCS(2), we obtain a tight factor-3 approximation.
3. For MCCS(k), we obtain a factor-($2k$-1) approximation.

We discuss the hardness and approximation algorithms for these problems in the following two sections and then conclude the paper in the last section.

2 Hardness of the MCCS Problem

In this section we prove that MCCS is NP-hard even when each color spans at most two points. We prove the NP-hardness of MCCS(2) by a reduction from Planar 3SAT [17], see Figure 2. The Planar 3SAT problem is equivalent to the 3SAT problem restricted to planar formulae.

Theorem 1. *MCCS(2) is NP-hard.*

Proof. We prove the hardness of MCCS(2) by a reduction from Planar 3SAT. Let ϕ be a Boolean formula in conjunctive normal form with n variables $x_1, \ldots, x_n$ in m clauses $\phi_1, \ldots, \phi_m$, each of size at most three. Given the planar embedding of ϕ, we take the following steps to construct a set of points S for of MCCS(2).

For each Boolean variable x_i in ϕ, let k_i^+ and k_i^- be the number of times x_i and $\overline{x}_i$ appears in ϕ respectively, and $k_i = \max\{k_i^+, k_i^-\}$. We use k_i chains labeled with $+$ and k_i chains labeled with $-$. (If $k_i^- < k_i = k_i^+$, then we just make sure $k_i - k_i^-$ chains labeled with $-$ do not connect to any clause; and vice versa. For convenience, we call

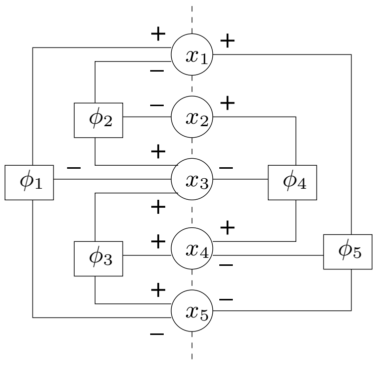


Fig. 2. An instance of planar 3SAT. The circles represent variables, the rectangles represent clauses, the $+$,$-$ on x_i denote the clause connects to literal x_i or $\overline{x}_i$ respectively.

such a chain 'dummy' chain as it does not affect the truth assignment.) These (non-dummy) chains are connected to some fixed points, each of a distinct color (which only appears once and must be selected). These fixed points, denoted by the empty circles in Figure 3, form a variable gadget. Note that the neighboring fixed points have a fixed distance of d_0. See Figure 3.

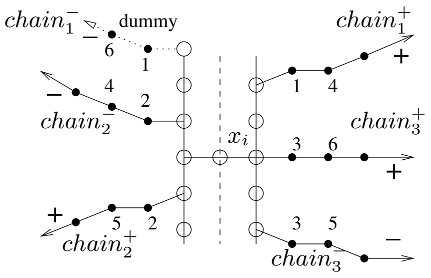


Fig. 3. The variable gadget where each number represents a color

Let the chains labeled with $+$ ($-$) around a variable x_i be sorted in counterclockwise order and let them be $chain_1^+, chain_2^+, ..., chain_{k_i}^+$ and $chain_1^-, chain_2^-, ..., chain_{k_i}^-$ respectively. Then, we use $2k_i$ points with k_i colors (each color spans two points). For each of these k_i colors, we put one point of the j-th color on $chain_i^+$ and the other point of the j-th color on $chain_i^-$. Each of these points is the first point of $2k_i$ chains respectively. This process is repeated until $j = k_i$. In Figure 3, these points correspond to the points labeled with 1, 2 and 3.

We use another $2k_i$ points with k_i colors where each color spans two points. For these k_i colors, we put one point of the j-th color on $chain_i^+$ and the other point of the j-th color on $chain_{i+1}^-$ (we take $chain_1^-$ = $chain_{k_i+1}^-$). This process is repeated until $j = k_i$. Each of these points is the second point of $2k_i$ chains respectively. The distance between the first two adjacent points on any of these $2k_i$ chains is set to be exactly d_0. In Figure 3, these points correspond to the points labeled with 4, 5 and 6.

For the other points on the chains, the sequence is not important. For any color, we just put one point on a chain with label $+$ and the other on a chain with label $-$, as long as there are not two points of the same color on a chain. Starting from the third point on each chain, the adjacent points might have a distance less than or equal to d_0. This will allow us to construct chains of different lengths. Of course, right before a chain reaches a clause, we need to perform something similar to the first two points on each chain. This will be discussed when we cover the clause gadget next.

The idea is that if we need to choose one point for each color to construct the variable x_i, all these points chosen need to be connected (via a communication range of $d_0/2$) to the fixed point of x_i. We either choose the point set on the chains with label $+$, or the point set on the chains with label $-$. The points in different variables have totally different colors. All the fixed points of variables are connected by adding some fixed points, see the dashed line in Figure 3.

For each clause $\phi_p = (x_i \vee x_j \vee \overline{x}_k)$ in ϕ, we add one fixed (clause) point and six points with three colors, two for each color. We try to connect the three chains (corresponding to the three literals in ϕ_p) to the fixed point as follows. We put two points with different colors at the end of each chain such that the three points next to the fixed (clause) point have different colors (e.g., 1,2 and 3 in Figure 4) and they are at distance d_0 to the fixed clause point. Then we connect three chains (in this example, chains with label $+$ for x_i and x_j and with label $-$ for x_k) by using three points in a permutation of these three colors such that the last two points on each chain are of different colors and the distance between them is d_0. See Figure 4. The unique design of the clause gadget makes sure that the fixed point of ϕ_p can only connect to exactly one variable of x_i, x_j, x_k.

Recall that for two intermediate points on a chain, their distance could be less than d_0. For two points p, q from two different chains, we make $d(p, q) > 2d_0$ to ensure that there are no edges between points from different chains.

As the fixed (clause) point for ϕ_p has to connect one of the fixed (variable) point of x_i, x_j, x_k, it is only possible when x_i is true, or x_j is true, or x_k is false. In fact, the clause point ϕ_p can only connect to one variable of x_i, x_j, x_k even if there are more than one literals making ϕ_p true.

Let S be the set of points hence constructed. We finally prove that the planar 3SAT instance ϕ is satisfiable if and only if there is a connected color-spanning $\frac{d_0}{2}$-Disk Graph of S.

"$\rightarrow$": If the planar 3SAT instance ϕ is satisfied, then each clause could connect to one variable. For each variable x_i, we either choose all the points on the chains labeled with $+$, or choose choose all the points on the chains labeled with $-$, which means we choose one point for each color. Let M be the points selected. All the variable points are connected through fixed points, then the $\frac{d_0}{2}$-Disk Graph on M is connected.

"$\leftarrow$": If there is a connected color-spanning $\frac{d_0}{2}$-Disk Graph on a subset of points of S, first notice that in our design of variable and clause gadgets, all the points chosen on the chains between variable x_i and the clause containing x_i or $\overline{x}_i$ must connect to the fixed point of variable x_i. Otherwise, the $\frac{d_0}{2}$-Disk Graph on the chosen points is not connected. According to the configuration of a variable gadget, we either choose the points on $chain_1^+, chain_2^+, ..., chain_{k_i}^+$ or the points on $chain_1^-, chain_2^-, ..., chain_{k_i}^-$.

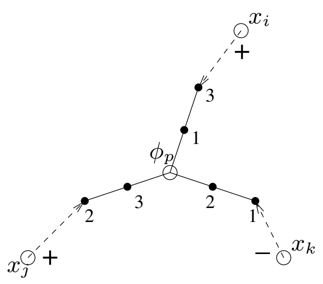


Fig. 4. Clause gadget for $\phi_p = (x_i \vee x_j \vee \overline{x}_k)$. Different numbers mean different colors.

For the $4k_i$ points to construct the first two points for all the variables, we either choose the points on the chains labeled $+$ or the points on the chains labeled $-$. Then we either choose all the immediate points connecting the chains labeled $+$ or the immediate points connecting the chains labeled $-$. (Otherwise, either a clause cannot be connected to any variable it contains, or the subset of points we choose does not span all the colors.) The first case represents the value **True** for this variable, and the second case represents the value **False**. As the fixed point of each clause connects to at least one variable, which means at least one literal in that clause is true, the instance ϕ is hence satisfied.

Therefore, the planar 3SAT instance ϕ is satisfiable if and only if there is a connected color-spanning $\frac{d_0}{2}$-Disk Graph of S. □

With our construction, we have in fact proved that MCCS(2) is NP-hard to approximate with a factor of $2 - \varepsilon$. We can strengthen the result by proving that MCCS(2) is NP-hard to approximate within a factor of $3 - \varepsilon$. We briefly summarize the necessary changes in the next theorem.

Theorem 2. *MCCS(2) is NP-hard to approximate within a factor of* $3 - \varepsilon$*, for some* $\varepsilon > 0$.

Proof. Omitted due to space limitation. □

In the next section, we present approximation algorithms for MCCS(2) and MCCS(k).

3 Approximation Algorithms for MCCS(k)

As a warm-up, we first discuss a special 2SAT instance which will be the basis of our approximation algorithm for MCCS(2). As we will see a bit later, it is a special case of bipartite graphs which always admit a perfect matching following Hall's Marriage Theorem [14]. But the 2SAT formulation is straightforward and is easier for implementation purpose.

Lemma 1. *Let 2SAT(1) be a special instance of 2SAT where a variable* x_i *and its negation* $\overline{x}_i$ *each appears at most once in the instance. Then 2SAT(1) always has a truth assignment.*

Proof. Let ϕ be a 2SAT(1) instance and the clauses are ϕ_1, $\phi_2, \cdots$. Each clause is composed of two literals, e.g., $\phi_j = (x_j \vee y_j)$. $(x_j \vee y_j)$ is equivalent to $\overline{x}_j \to y_j$ and $\overline{y}_j \to x_j$. So we have a directed graph $D(\phi)$ on all the literals. If ϕ is not satisfiable, then suppose there is a path $x_i \to x_2 \to \cdots x_\ell \to \overline{x}_i$ in $D(\phi)$ which does not contain nodes x_k and $\overline{x}_k$ between x_i and $\overline{x}_i$ (otherwise, we take a proof for x_k). Then, the clause corresponding to the path is $(\overline{x}_i \vee x_2) \wedge \cdots \wedge (\overline{x}_\ell \vee \overline{x}_i)$. Hence, either the literal $\overline{x}_i$ appears twice in ϕ (a contradiction) or $x_\ell = \overline{x}_2$. If $x_\ell = \overline{x}_2$, then there is a path from x_2 to $\overline{x}_2$ between x_i and $\overline{x}_i$, again a contradiction to the assumption. □

3.1 Approximation Algorithm for MCCS(2)

Given an $\frac{r}{2}$-Disk Graph G with n points and m colors, each node of G is painted with one color, we want to choose a set T of m nodes (one node for each color) from G. We define a graph $H =< T, E' >$, where there is an edge $(u, v) \in E'$ for two nodes $u, v \in T$ if there is a path between u and v of length at most k in G. If H is connected for some value k, we say that H is a $(k-1)$*-hop color-spanning subgraph* of G. In Figure 5, if we select H as node 1 and the remaining doubly labeled nodes from 2 to 6, then H is a 1-hop color-spanning subgraph of G. We now prove the following lemma regarding MCCS(2).

Lemma 2. *Given an $\frac{r}{2}$-Disk Graph G with m colors and each color spans at most two points, if there exists a connected component of G which contains all the m colors, then there is a 2-hop color-spanning subgraph H of G.*

Proof. If this connected component only contains exactly one point z of certain color, then we say z is a *fixed* point. Obviously, a fixed point must be selected to form any color-spanning subgraph. We also do some preprocessing by removing any edge between two nodes of the same color — as such an edge cannot be in any optimal solution. As there exists a connected component of G whose nodes contain all the m colors, we perform a depth-first search on this connected component from a fixed node (point) of G and if there is no fixed point then start with any node. In the searching process, we build a disjoint set of groups, each containing an edge of G, as follows. Let a be the current node which has not been completely explored (see [8]) and let b a neighbor of a in G. If both a and b are not fixed points, and neither a nor b is already in some group, then we build a new group $\{a, b\}$.

Suppose that there are a total of m_1 groups and each group has two points of different colors, hence there are m_2 colors in the m_1 groups with $m_2 \geq m_1$. See Figure 5. In the m_1 groups, if a color paints only one point, then we simply choose that point for H. If a color spans two points in the m_1 groups, we need to choose one for H. We use x_i and $\overline{x}_i$ to denote the two points of color c_i respectively, and each group G_t $(1 \leq t \leq m_1)$ containing two points of color c_i and c_j can be expressed as a clause like $(x_i \vee \overline{x}_j))$. x_i (resp. $\overline{x}_j$) is assigned true when the point of color c_i (resp. c_j) in the group G_t is chosen. Then, the m_1 groups can be expressed as an instance I of 2SAT(1).

By Lemma 1, the above 2SAT(1) instance always has a truth assignment. The truth assignment gives us the selection of the corresponding points for H. If a color spans two points in the connected component but these points never appear in the m_1 groups, then just choose any one of the points for H. Recall that if a color contains only one point in the connected component, we choose that fixed point at the first place for H. Hence we choose m points H from G to have m distinct colors.

Within this connected component of G which contains m distinct colors, from the above construction, it can be seen that between two groups there can be either an edge connecting two nodes from the two groups, or the two groups are connected by a sequence of fixed points. Let these two groups be $G_i = \{a_i, b_i\}$ and $G_j = \{a_j, b_j\}$ respectively. In the worst case, we select one point each from them (say, a_i and b_j) for H, leaving the other two as hops to maintain connectivity in G; i.e., $a_i \rightarrow b_i \rightarrow a_j \rightarrow b_j$. Hence, there are at most three edges (or two hops) in G connecting points in H whose corresponding groups are adjacent.

For any non-fixed point p selected for H which does not belong to any group, p is either adjacent to some fixed point or is adjacent to a point in some group G_t. Otherwise p and one of its neighbors would be forming a new group. Hence, there are at most two edges (or one hop) between p and its nearest point in H.

In summary, if there exists a connected component of G which contains all the m colors, then there is a 2-hop color-spanning subgraph H of G. □

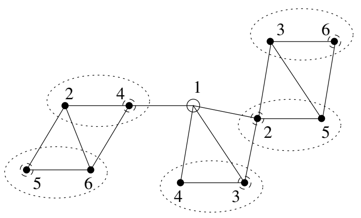


Fig. 5. A connected component of graph G which is divide into groups by DFS and each group has just two points.

The above lemma implies that for nodes in H, if we increase their communication range to $\frac{3r}{2}$, then H will be connected even if all other nodes in $G - H$ are deleted.

Theorem 3. *There is a factor-3 approximation for MCCS(2).*

Proof. It is easy to see that the optimal solution value r^* must be the distance between a pair of points of S which are of different colors. We sort the distances between all pairs of n points of S, and let $d_1, d_2, d_3, ..., d_q$ be the sorted sequence. We try each of d_i as r, for $i = 1, 2, ..., q$, and build the corresponding $\frac{r}{2}$-Disk Graph $G(r)$. Suppose that $G(r)$ contains a connected component which contains of all the m colors the first time when the value of r is increased to d_j, then the optimal solution value r^* satisfies $r^* \geq d_j$. The reason is that if it is not the case, the number of colors of any connected component of $G(r)$, $r < d_j$, is less than m; hence, there are at least two colors whose corresponding points belong to two different components (which is at least d_j distance away). Therefore, when $r < d_j$, it is impossible to find a color-spanning subgraph of $G(r)$ which is connected.

Following Lemma 2, we can compute a 2-hop color-spanning subgraph $H(d_j)$ of $G(d_j)$. In other words, distance between two adjacent points in $H(d_j)$ is at most $3d_j$, as d_j is the maximum length of any edge in a connect component of $G(d_j)$ which contains m colors. This means that we obtain an approximation whose solution value APP satisfies

$$APP \leq 3d_j \leq 3r^*.$$

We finally analyze the time complexity of the algorithm. Computing and sorting $O(n^2)$ distances takes $O(n^2 \log n)$ time. Each time the value of r is increased from r', we either add an edge into a connected component of $G(r')$ (which takes $O(1)$ time) or merge two connected components of $G(r')$ into one (which takes $O(\alpha(n))$ on average — if we use the standard union-find data structure as some auxiliary structure to test whether two elements lie in the same connected component [8,22].) As there are $O(n^2)$ edges and $O(n)$ merges, the total cost is $O(n^2)$. A connected component contains m colors only when there are at least m points in it, that means the graph $G(d_j)$ has at most two connected components satisfying this condition. Hence the time to decide if a connected component contains $m = \Theta(n)$ colors takes $O(n)$ time. When we have a connected component satisfying the condition, the depth-first search, and solving the resulting 2SAT(1) instance takes $O(n)$ time.

Hence the total time complexity is $O(n^2 \log n)$, and the space complexity is $O(n^2)$. □

3.2 Approximation Algorithm for MCCS(k)

For the more general case when each color spans at most k points, we use a similar method as in the previous section until the first time we obtain a $\frac{d_j}{2}$-Disk Graph $G(d_j)$ such that it contains a connected component which contains all the m colors. On any such connected component, we can use the depth-first search (or other method, say a spanning tree) to divide the component into (connected) groups, each containing exactly k points. Then, we choose at least one point from each group (which is proven to be always possible in the next lemma), to form the color-spanning subgraph H. Since in the worst case two points selected are from two neighboring groups, which could be 2(k-1) hops away (or, $2k$-1 edges away), we can give each point selected for H a communication radius of ($2k$-1)d_j to make H connected. Hence, we obtain a factor-($2k$-1) approximation for MCCS(k).

Lemma 3. *In a connected component of $G(d_j)$ which contain all the m colors, if there are g groups of points containing the m colors ($g \leq m$), each group has exactly k points, and each color spans at most k points, then we can always choose one point for each color such that each group has at least one point chosen.*

Proof. We construct a bipartite graph (U, V, E): U denotes the g groups $\{G_1, G_2, ...,$ $G_g\}$, V denotes the m colors $\{c_1, c_2, ..., c_m\}$, and there is an edge between G_i and c_j iff the group G_i contains at least one point of color c_j. Following Hall's Marriage Theorem [14], which, in this setting, states that there is a perfect matching for the bipartite graphs (U, V, E) iff the degree of nodes in U's are at least m, there is a perfect matching where all nodes in U (or groups) will be in a matching. Then, the lemma follows immediately. □

We comment that the 2SAT(1) instance we covered previously is a special case of the bipartite graph we have just discussed. Combined with the previous discussions, it is easily seen that this is gives us a polynomial time approximation. In addition, as the maximum matching algorithm takes $O(n^{5/2})$ time [15] but can be improved to $O(n^2 \log k) = O(n^2)$ time for regular bipartite graphs [7], so the overall running time of this algorithm remains to be $O(n^2 \log n)$. (We comment that with a randomized solution the perfect matching can be computed in $O(n \log n)$ time [13], but it will not change the overall running time of our algorithm.)

Theorem 4. *There is a factor-($2k$-1) approximation for MCCS(k).*

4 Concluding Remarks

We give tight approximation bounds for the Minimum Connected Color-Spanning Set problem, which arises in wireless sensor networks. When k is big, the $2k$-1 factor for MCCS(k) might not be efficient enough. So an interesting question is whether a constant factor approximation can be obtained for MCCS.

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristan, V.: The farthest color Voronoi diagram and related problems. In: Proceedings of the 17th European Workshop on Computational Geometry (EWCG 2001), pp. 113–116 (2001)
2. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristan, V.: Smallest Color-Spanning Objects. In: Proc. 9th Annu. European Sympos. Algorithms, pp. 278–289 (2001)
3. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Computing 2(1), 46–55 (2003)
4. Cheema, M.A., Lin, X., Wang, W., Zhang, W., Pei, J.: Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data. IEEE Trans. Knowl. Data Eng. 22(4), 550–564 (2010)
5. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments, knowledge and data engineering. IEEE Transactions on Knowledge and Data Engineering 16(9), 1112–1127 (2004)
6. Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving user location privacy in mobile data management infrastrutures. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 393–412. Springer, Heidelberg (2006)
7. Cole, R., Ost, K., Schirra, S.: On edge coloring bipartite graphs. Combinatorica 21(1), 5–12 (2001)
8. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press (2001)
9. Das, S., Goswani, P.P., Nandy, S.C.: Smallest color-spanning object revised. International Journal of Computational Geometry and Applications 19(5), 457–478 (2009)
10. Efrat, A., Fekete, S.P., Gaddehosur, P.R., Mitchell, J.S.B., Polishchuk, V., Suomela, J.: Improved approximation algorithms for relay placement. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 356–367. Springer, Heidelberg (2008)
11. Fleischer, R., Xu, X.: Computing Minimum Diameter Color-Spanning Sets. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 285–292. Springer, Heidelberg (2010)

12. Gedik, B., Liu, L.: A customizable k-anonymity model for protecting location privacy. In: Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005), pp. 620–629 (2005)
13. Goel, A., Kapralov, M., Khanna, S.: Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. In: Proceedings of STOC 2010, pp. 39–46 (2010)
14. Hall, P.: On representatives of subsets. J. London Math. Soc. 10(1), 26–30 (1935)
15. Hopcroft, J., Karp, R.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2(4), 225–231 (1973)
16. Ju, W., Fan, C., Luo, J., Zhu, B., Daescu, O.: On Some Geometric Problems of Color-Spanning Sets. J. of Combinatorial Optimization 26(2), 266–283 (2013)
17. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. 11(2), 329–343 (1982)
18. Mumey, B., Spendlove, K., Zhu, B.: Extending the lifetime of a WSN by partial covers. In: Proceedings of IEEE ICC 2013 (2013)
19. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic Skylines on Uncertain Data. In: Proc. of VLDB 2007, pp. 15–26 (2007)
20. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-objects representations. In: Güting, R.H., Papadias, D., Lochovsky, F.H. (eds.) SSD 1999. LNCS, vol. 1651, pp. 111–131. Springer, Heidelberg (1999)
21. Sistla, P.A., Wolfson, O., Chamberlain, S., Dao, S.: Querying the uncertain position of moving objects. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Temporal Databases-Research and Practice. LNCS, vol. 1399, pp. 310–337. Springer, Heidelberg (1998)
22. Tarjan, R.: Efficiency of a good but not linear set union algorithm. J. of ACM 22(2), 215–225 (1975)
23. Yang, Y., Lin, M., Xu, J., Xie, Y.: Minimum spanning tree with neighborhoods. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 306–316. Springer, Heidelberg (2007)
24. Yuen, S.-M., Tao, Y., Xiao, X., Pei, J., Zhang, D.: Superseding Nearest Neighbor Search on Uncertain Spatial Databases. IEEE Trans. Knowl. Data Eng. 22(7), 1041–1055 (2010)
25. Zhang, D., Chee, Y.M., Mondal, A., Tung, A.K.H., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE 2009), pp. 688–699 (2009)
26. Zhang, H., Hou, J.C.: Maintaining sensing coverage and connectivity in large sensor networks. Ad Hoc & Sensor Wireless Networks, an Intl J. 1, 89–124 (2005)
27. Zhang, H., Nixon, P., Dobson, S.: Partial coverage in homological sensor networks. In: Proceedings of the 5th IEEE Intl. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2009), pp. 42–47 (2009)

The Train Delivery Problem Revisited

Jing Chen[1], He Guo[2], Xin Han[2], and Kazuo Iwama[1]

[1] Graduate School of Informatics, Kyoto University
chen@algo.cce.i.kyoto-u.ac.jp,
iwama@kuis.kyoto-u.ac.jp
[2] Software School, Dalian University of Technology
{hanxin,guohe}@dlut.edu.cn

Abstract. In this paper, we study the *train delivery* problem which is a generalization of the bin packing problem, and is also equivalent to a one dimensional version of the vehicle routing problem with unsplittable demands. We give an APTAS for the general problem with time complexity $O(n^{O(\epsilon^{-4})})$, which is better than the previous one $O(n^{(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}})$, where n is the number of input elements.

1 Introduction

The train delivery problem was first studied by Das, Mathieu and Mozes in 2010, and has many applications in the real world [3]. In the problem, there are trains with capacity W and a set of products, which are called customers, each customer is associated with (s, p), where s is the size and p is the position. A set of products with a total size at most W can be shipped together, however the cost of the shipment is dominated by the largest position value of products in the train. We are asked to ship all the products and minimize the total cost occurred. If all the positions are equal to one, the problem is degenerated to one dimensional bin packing problem [1].

Formally, the train delivery problem(TDP) is defined as below: given a positive capacity W(positive real number) and a set $S = \{(s_1, p_1), (s_2, p_2), ..., (s_n, p_n)\}$ of n customers, where all s_i and p_i are positive numbers, we are asked to partition S into subsets $\{S_j\}$(train tours) to minimize

$$\sum_j \max_{i \in S_j} p_i \quad \textit{subject to} \quad \forall j \sum_{i \in S_j} s_i \le W.$$

Related Work: Das, Mathieu and Mozes [3] first mentioned that the problem does not admit a polynomial time approximation algorithm with an approximation factor strictly less than 1.5, then designed an asymptotic polynomial time approximation scheme (APTAS), under the condition: the largest position is arbitrarily small compared with the optimal value, finally they purposed a polynomial time approximation scheme (PTAS) for the case where W is polynomial in the input size n.

Our Contribution: In this paper, we improve the time complexities of schemes by Das, Mathieu and Mozes [3]. In details,

1. if $\max_i p_i = O(\epsilon)OPT$, we give APTAS in time $O(n^{O(\epsilon^{-4})})$, which is better than the one $O(n^{(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}})$.
2. If W is polynomial in n, the time complexity of the PTAS given in [3] is improved from $O(W^{e^{O(\frac{1}{\epsilon})}}) + O(n^{(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}})$ to $O(W^{e^{O(\frac{1}{\epsilon})}}) + O(n^{O(\epsilon^{-4})})$.

Compared with one dimensional bin packing problem, the difficulty in our problem is that: customers with small sizes cannot be ignored, since they may have big positions. Das, Mathieu and Mozes [3] also mentioned that the approaches for bin packing problem cannot be applied to the TDP *directly.*

To design an APTAS, Das, Mathieu and Mozes [3] partition customers into disjoint regions based on their positions, each region has only a constant number of locations containing customers. For each location the customers are grouped into constant different sets, each set has a distinct size, so there are at most a constant number of different sizes in one region. Then they use an exhaustive enumeration of all solutions of the large customers. Finally, for each solution they insert small elements greedily to it.

In our paper, combining the techniques in layered graph scheme given by [11](for scheduling problem), also used in [10] (for generalized cost variable sized bin packing), we design a layered graph with a source vertex s and a terminal vertex t for a rounded instance, such that any path from s to t corresponds a feasible solution of packing products into trains, and each edge (u, v) in the graph corresponds to a pattern of one train, so the length of (u, v) is equal to the cost by the train, and vertex u contains the information of unpacked products. If there are only large products in the input, then the problem is done, since finding the shortest path is enough. The problem is that: how to pack small products in the train? To attack the problem, in each train, we keep some space for small products, and there are constant distinct possibilities for the space. Sometimes we are allowed that the train is overflowed, i.e., the total size in the train is larger than W due to accepting small customers. In the graph we constructed, we make sure there are enough space for the small customerss, but we do not give a packing solution for small customers. To obtain APTAS, we first find a shortest path in the graph, which is the lower bound of optimal solution, since we allowed a train to be overflowed by the space prepared for small customers. We first make sure all small customers can be packed into the space prepared for them, then we take the small customers out the trains that overflowed and pack them greedily into some extra trains. We find the cost by extra trains is at most $O(\epsilon)OPT + O(p_{max})$, where p_{max} is the maximum position of all the customers. This approach leads to an APTAS, which is different from the one in [3]. Throughout this paper we denote by ϵ a fixed positive constant such that $\epsilon < \frac{1}{100}$ and $\frac{1}{\epsilon}$ is an integer.

2 Rounding Up Positions and Sizes

To design an APTAS, the first step is to round the input such that the input is structured and the lost by rounding is not much significant. In this section we first round positions of products up to $\frac{p_{max}}{(1+\epsilon)^i}$ for some i, where p_{max} is the maximum position of all the products. And we guarantee that we enlarge the optimal solution within a factor $(1+\epsilon)$. Then we round up large sizes of products into a constant number of different sizes by the techniques used in bin packing problem, and also guarantee that the rounding does not affect the optimal solution too much.

2.1 Rounding Up Positions

We sort the customers(customer i be denoted as (s_i, p_i), with size s_i and position p_i) by their positions, such that $p_1 \geq p_2 \geq ... \geq p_n > 0$, let $p_{n+1} = 0$. We traverse the input from p_1 to p_n and round up p_i to $\bar{p}_i$ for $i \geq 1$, such that:

$$\bar{p}_i = \frac{p_1}{(1+\epsilon)^k},$$

where k is the maximum integer such that:

$$\frac{p_1}{(1+\epsilon)^{k+1}} < p_i \leq \frac{p_1}{(1+\epsilon)^k}.$$

Lemma 1. *Let L be an input list, and L' be the list generated by rounding up as described above, then $OPT(L') \leq (1+\epsilon) \cdot OPT(L)$, where $OPT(L')$ is the optimum cost of L' and $OPT(L)$ is the optimum cost of L.*

The proof is left in the full version.

2.2 Grouping *Large* Elements

Definition 1. *A customer of size s_j is large if $s_j \geq \epsilon W$, else it is small.*

Let L be the input after rounding up positions. Let $T = \{\mathbb{P}_1, \mathbb{P}_2, ..., \mathbb{P}_r\}$ be the set of positions from L, where $\mathbb{P}_1 \leq \mathbb{P}_2 \leq ... \leq \mathbb{P}_r$.

Then $r = |T|$ denotes the number of different positions in L, we have $r \leq n$.

Next we partition input list L into subsets according to sizes of the customers.

Definition 2. *Let $\mathcal{L}_k$ be the set of large customers with position $\mathbb{P}_k$. If $|\mathcal{L}_k| < \frac{1}{\epsilon^2}$ we call $\mathcal{L}_k$ as a thin class, else if $|\mathcal{L}_k| \geq \frac{1}{\epsilon^2}$ we call it as a fat class.*

Given a class $\mathcal{L}_k$ we want to group $\mathcal{L}_k$ into $\frac{1}{\epsilon^2}$ sub-groups, assume that $\frac{1}{\epsilon}$ is an integer. We have two cases.

Case 1: If $\mathcal{L}_k$ is a *thin* class, that is $|\mathcal{L}_k| < \frac{1}{\epsilon^2}$. We sort the customers of $\mathcal{L}_k$ by their sizes(by decreasing order) and partition them into $|L_k|$ parts, each part has one customer, that is $|S_1^k| = 1$, $|S_2^k| = 1$, ..., $|S_{|L_k|}^k| = 1$, $|S_{|L_k|+1}| = 0$, ..., and the size of the customer of S_i^k is larger than or equal to the size of the customer of S_j^k(if exists) for $i < j$.

Case 2: If $\mathcal{L}_k$ is a *fat* class, that is $|\mathcal{L}_k| \geq \frac{1}{\epsilon^2}$. Let $|\mathcal{L}_k| = n_k$. We sort the customers $c_1^k, c_2^k, ..., c_{n_k}^k$ of $\mathcal{L}_k$ according to their sizes. Denote the size of the customer c_j^k by s_j^k, and without loss of generality we assume that $s_1^k \geq s_2^k \geq ... \geq s_{n_k}^k$. We partition $\mathcal{L}_k$ into $\frac{1}{\epsilon^2}$ subsets denoted by S_1^k, S_2^k, ..., S_{1/ϵ^2}^k. The partition is defined by the following two conditions: i) $|S_p^k| = \lceil n_k\epsilon^2 \rceil$ or $\lfloor n_k\epsilon^2 \rfloor$, ii) and for all $p, q \geq 1$, if $p < q$ then $|S_p^k| \geq |S_q^k|$(note that $|S_1^k| = \lceil n_k\epsilon^2 \rceil$), i.e., we partition $\mathcal{L}_k$ to approximately equal size sets, refer to Fig. 1.

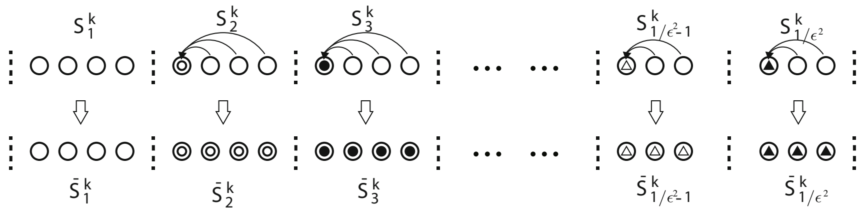


Fig. 1. An example of grouping *fat* class, in this case $\lceil n_k\epsilon^2 \rceil = 4$, $\lfloor n_k\epsilon^2 \rfloor = 3$

Set S_j^k contains the largest elements from $\mathcal{L}_k \backslash (S_1^k \cup S_2^k \cup ... \cup S_{j-1}^k)$. For all k and $j \geq 2$, we round up the size of all customers in S_j^k to the size of the largest element in S_j^k, the set of these rounded customers is denoted by $\bar{S}_j^k$, and define $\bar{S}_1^k = S_1^k$. We have the following lemma, and its proof is in the full version.

Lemma 2. *Let $\mathcal{S}_1 = \cup_{\mathcal{L}_k:fat} S_1^k$, the total cost of assigning each customer of $\mathcal{S}_1$ into a dedicated train is at most $2\epsilon \cdot OPT$, where OPT is the cost of an optimum solution.*

3 An Asymptotic Polynomial Time Approximation Scheme

Our scheme is based on construction of a layered graph, the path in this graph only have *large* elements actually packed, and there are slots of size ϵW prepared for packing back *small* elements in the path, we designed an algorithm(algorithm 1) to packing back the *small* elements, which we proved can assure the additional cost is within $O(\epsilon)$ times the cost of the path. We modified an optimal *small expanse* solution which is defined below, and showed that there is a corresponding path in our graph, then we find the shortest path in the graph, whose cost is definitely smaller than this path of the modified solution, and the results of applying algorithm 1 on the shortest path is a $1 + O(\epsilon)$ solution of the TDL problem.

Definition 3. *We say that a solution is small expanse: if in the solution for every pair of a train of cost p_i and a set of customers S packed in the train we have either $s_j \leq \epsilon W$ or $p_j \geq \epsilon p_i$ for $\forall j \in S$.*

In a *small expanse* solution, a customer in the train has small size or the position of the customer differs from the cost of the train by a constant factor. Then we focus on analyzing *small expanse* solutions and have the following lemma (its proof is left in the full version).

Lemma 3. *There exists a small expanse solution with cost at most* $(1+2\epsilon)OPT(L)$, *where* $OPT(L)$ *is the optimal solution of input* L.

3.1 Patterns in One Stage

Given an input L, after rounding up positions and large sizes, we have r distinct positions in L, denoted by $T = \{\mathbb{P}_1, \mathbb{P}_2, ..., \mathbb{P}_r\}$. Then our layered directed graph $G = (V, E)$ is composed of $r+1$ stages, and each stage is formed by a consecutive set of $n+1$ layers, in each layer there are a set of vertices, which will be defined later. The i-th stage corresponds to decisions regarding the covering(packing) of customers by trains of cost $\mathbb{P}_i$. Each edge connects a vertex of one layer to a vertex of the consecutive layer, where the layers are ordered so that first there are $n+1$ layers of stage $r+1$, then the layers of stage r, and so on up to the layers of stage 1. We add to G one additional vertex denoted by t, which is defined to be the final vertex.

For stage i, we define

$$B_i = \{k : \mathbb{P}_k \in [\epsilon\mathbb{P}_i, \mathbb{P}_i]\}.$$

A *pattern* of stage i corresponds to a packing of a train of cost $\mathbb{P}_i$ with elements in $\cup_{k\in B_i, j\geq 1}\bar{S}_j^k$. In a *pattern* we keep $\epsilon \cdot W \cdot \hat{n}_{slot}$ space for *small* elements, where $\hat{n}_{slot}$ is an integer. Formally, a *pattern* of stage i is defined as below:

$$(\hat{n}_{slot}, (\hat{n}_j^k)_{k\in B_i, j\geq 1}),$$

subject to

$$\begin{aligned} W \geq &\quad \textstyle\sum_{k\in B_i, j\geq 1} \hat{n}_j^k \cdot \bar{s}_j^k, \\ (1+\epsilon)W \geq &\quad \textstyle\sum_{k\in B_i, j\geq 1} \hat{n}_j^k \cdot \bar{s}_j^k + \epsilon W \cdot \hat{n}_{slot}, \end{aligned}$$

where $\hat{n}_j^k$ is the number of elements from $\bar{S}_j^k (\bar{S}_j^k = S_j^k$ if $\mathcal{L}_k$ is a *thin* class), $\bar{s}_j^k$ is the size of $\bar{S}_j^k$. Since elements from $\bar{S}_1^k$ of *fat* class are not considered in the *pattern*, we set $\hat{n}_1^k = 0$. Note that the total size in a *pattern* may overflow to $W + \epsilon \cdot W$. Actually, when we pack the real *small* customers back into the train, if it is overflowed, we use extra trains to handle the overflowed *small* customers. We have following lemma, its proof is left in the full version.

Lemma 4. *The number of possible patterns of stage* i *is* $O((\frac{1}{\epsilon}+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}+2})$.

3.2 Constructing Vertices of the Graph

Next we construct vertices of stage i in the graph. Each layer in the same stage has the same set of vertices. Roughly speaking, a vertex corresponds to elements unpacked so far. Each vertex u of stage i in the graph is associated with information *tag* which contains three parts:

$$tag =< B_i, n_{slot}(u), n_j^k(u) >,$$

where $k \in B_i, j \geq 1$. The first part of the information *tag* is the set B_i.

The second part indicates that the total size of *small* elements unpacked at this stage is at most $n_{slot}(u) \cdot \epsilon W$, where $n_{slot}(u)$ is an integer in $[0, n]$.

The third part contains information on the number $n_j^k(u)$ of elements from $\bar{S}_j^k$ that still needs to be packed. If $\mathcal{L}_k$ is a *fat* class, we set $n_1^k(u) = 0$.

Lemma 5. *The number of vertices in the graph is at most* $2^{\log_{1+\epsilon}\frac{1}{\epsilon}+1} \cdot (n+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}+3}$, *that is polynomial in the input size.*

Proof. We first analyze the maximum number of vertices that each stage can have. Consider stage i, let $\mathcal{G}_i$ be the set of all possibilities of information *tag* of vertex of stage i:

$$\mathcal{G}_i = \{g | g\text{: an information } tag \text{ of stage } i\ \}. \tag{1}$$

We consider the possible values of the information *tag* of vertices. Set B_i is identical for all of the vertices in this stage, the second part $n_{slot}(u)$ of the vertex information *tag* is an integer in the range $[0, n]$, thus the number of possible values of $n_{slot}(u)$ is at most $n + 1$,

and $n_j^k(u)$ is an integer in the range $[0, n_i]$, which are at most $n+1$ possibilities.

For the pairs of indices k and j, by definition $\mathbb{P}_k \in [\epsilon\mathbb{P}_i, \mathbb{P}_i]$ and $j = 2, 3, ..., \frac{1}{\epsilon^2}$, and by Lemma 1 there at most $1 + \log_{1+\epsilon}\frac{1}{\epsilon}$ values that k can have. For $j = 1$, $n_1^k(u)$ is an integer in the range $[0, 1]$, i.e., there are only 2 possibilities. Therefore, the number of possibilities for the third part of the information *tag* is at most $2^{\log_{1+\epsilon}\frac{1}{\epsilon}+1} \cdot (n+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}}$.

Therefore the maximum number of vertices of each stage is at most $(n+1) \cdot 2^{\log_{1+\epsilon}\frac{1}{\epsilon}+1} \cdot (n+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}}$, the number of veritecs in the graph is therefore at most $(r+1) \cdot (n+1)^2 \cdot 2^{\log_{1+\epsilon}\frac{1}{\epsilon}+1} \cdot (n+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}} \leq 2^{\log_{1+\epsilon}\frac{1}{\epsilon}+1} \cdot (n+1)^{\frac{\log_{1+\epsilon}\frac{1}{\epsilon}+1}{\epsilon^2}+3}$. $\square$

As showed in the proof of lemma 5, there is a finite set $\mathcal{G}_i$ of information *tag* for each stage i, we construct each layer to have a vertex for each *tag* in $\mathcal{G}_i$, the number of vertices of the layer is thus $|\mathcal{G}_i|$.

First there is a vertex s of stage $r + 1$, then we construct the $n + 1$ layers of stage r as above, and so on until the $n + 1$ layers of stage 1 and finally we add one final vertex t.

3.3 Connecting Feasible Edges between Vertices

There are four types of edges. The first two types of edges connect two vertices of two consecutive layers in the same stage. The last two types of edges connect vertices from two consecutive stages.

1. The first type of an edge connects two vertices from two consecutive layers in stage i. It corresponds to a *pattern* of stage i.

Let u and v be vertices in stage i, and v is in the consecutive layer of u, then there is a edge of type one between u and v if the following conditions hold:

$$0 \le \hat{n}_{slot} = n_{slot}(u) - n_{slot}(v), \tag{2}$$

$$0 \le \hat{n}_j^k = n_j^k(u) - n_j^k(v),\ k \in B_i,\ j \ge 1, \tag{3}$$

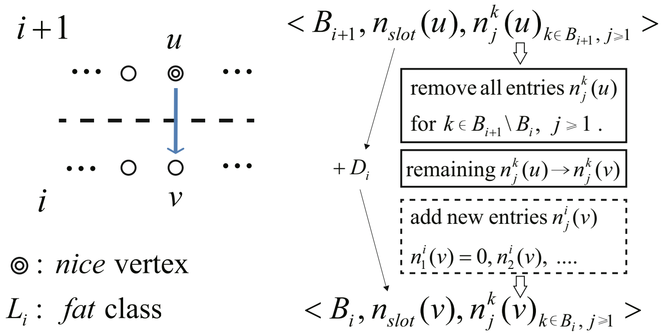
$$0 \le W - \sum_{k\in B_i, j\ge 1} \hat{n}_j^k \cdot \bar{s}_j^k, \tag{4}$$

$$\hat{n}_{slot} \le \lceil \tfrac{W-\sum_{k\in B_i,j\ge 1}\hat{n}_j^k\cdot\bar{s}_j^k}{\epsilon W}\rceil, \tag{5}$$

where $\bar{s}_j^k$ be the size of elements in $\bar{S}_j^k$. That is if the difference of information tags of u and v forms a feasible *pattern* then there is a edge of type one.

2. The second type of an edge connects two vertices in consecutive layers in the same stage with equal information *tag*. Such an edge has a zero cost. Such an edge corresponds to skip a layer(train).

3. The third type of an edge is about the connection of two consecutive stages. First let V_{i+1} denote the set of all vertices of the last layer of stage $i+1$. A vertex $u \in V_{i+1}$ is called *nice* if it satisfies the following condition: for all $k \in B_{i+1}\backslash B_i$, $n_j^k(u) = 0$, $j = 1, ..., \frac{1}{\epsilon^2}$. The third type of an edge connects only *nice* vertices of the last layer of a stage $i+1$ to the vertices of the first layer of the consecutive stage i. Let (u, v) be such an edge where u is *nice*. Then vertex v of stage i satisfies the following conditions: i) $n_{slot}(v) = \lceil \frac{n_{slot}(u)\cdot\epsilon W + D_i}{\epsilon W}\rceil$, where D_i is the total size of *small* elements of position $\mathbb{P}_i$; ii) for all $k \in B_{i+1} \cap B_i$, we have $n_j^k(v) = n_j^k(u)$ for all $j = 1, ..., \frac{1}{\epsilon^2}$; iii) for all $k \in B_i\backslash B_{i+1}(B_i\backslash B_{i+1} = \{i\})$, we have $n_j^k(v) = |\bar{S}_j^k|$ for all $j = 1, ..., \frac{1}{\epsilon^2}$; if $\mathcal{L}_k$ is a *fat* class, we set $n_1^k(v) = 0$.



$i+1$
u
i
v
◎ : *nice* vertex
L_i : *fat* class
$< B_{i+1}, n_{slot}(u), n_j^k(u)_{k\in B_{i+1}, j\geqslant 1} >$
remove all entries $n_j^k(u)$ for $k \in B_{i+1}\backslash B_i,\ j \geqslant 1$.
remaining $n_j^k(u) \to n_j^k(v)$
$+D_i$
add new entries $n_j^i(v)$
$n_1^i(v) = 0, n_2^i(v), ...$
$< B_i, n_{slot}(v), n_j^k(v)_{k\in B_i, j\geqslant 1} >$

Fig. 2. Edge (u, v) of type 3 where the class of i is *fat*

The cost of the edge (u, v) is $\sum_{k\in B_{i+1}\setminus B_i, L_k: fat} |\bar{S}_1^k| \cdot \mathbb{P}_k$, this cost reflects the cost of assigning each element of $\bar{S}_1^k$ ($\mathcal{L}_k$ is a *fat* class) to a dedicated train of cost $\mathbb{P}_k$. We charge this cost to an edge of type 3 or type 4, at the time when k stops appearing in the first part of the information *tag*.

4. The last type of edge connects vertex u of the last layer of stage 1 to t. If $n_j^k(u) = 0$ and $n_{slot}(u) = 0$ for all j, k. The cost of such an edge is $\sum_{k\in B_1, \mathcal{L}_k: fat} |\bar{S}_1^k| \cdot \mathbb{P}_k$.

3.4 Constucting a Feasible Solution from a Path of the Graph

In our scheme we first finds the minimum cost route(we use the word route synonymous with path) $\mathcal{R}$ from s to t. Then we construct a feasible solution based on the path found by the procedure as below.

If the path uses an edge connecting (u, v) where both of them belong to a common stage i,

then this edge corresponds to a pattern $(\hat{n}_{slot}, (\hat{n}_j^k)_{k\in B_i, j\geq 1})$, where $\hat{n}_{slot} = n_{slot}(u) - n_{slot}(v)$ and $\hat{n}_j^k = n_j^k(u) - n_j^k(v),\ k \in B_i,\ j \geq 1$. We use a train of cost $\mathbb{P}_i$ to allocate it with exactly $\hat{n}_j^k$ elements of $\bar{S}_j^k$ for all $k \in B_i$ and $j \geq 1$, reserve a space of size $\hat{n}_{slot} \cdot \epsilon W$ for small items. We apply this for all edges of the first type that $\mathcal{R}$ uses. The second type of edges in $\mathcal{R}$ do not affect the solution.

Next assume that $\mathcal{R}$ uses an edge (u, v) of the third type where u belongs to the $(i + 1)$-th stage and v belongs to stage i.

For all $k \in B_{i+1}\setminus B_i$, $\mathcal{L}_k$ is *fat*, we open $|\bar{S}_1^k|$ trains of cost $\mathbb{P}_k$ and allocate them each with one element of $\bar{S}_1^k$. Consider the edge of the last type that $\mathcal{R}$ uses. For all $k \in B_1$, $\mathcal{L}_k$ is *fat*, we use $|\bar{S}_1^k|$ trains of cost $\mathbb{P}_k$ and allocate them each with one element of $\bar{S}_1^k$.

The total cost of trains that we used is exactly the cost of $\mathcal{R}$. We next consider the non-allocated *small* elements(for which the path $\mathcal{R}$ reserves space by $\hat{n}_{slot}$). We sort these elements in a non-increasing order of positions.

For each edge that contains at least one slot for *small* elements, we give one additional slot ϵW for it, the total size of *large* element packed in a train plus the space of slots may overflow to at most $W + 2\epsilon W$. We pack *small* elements into the space denote by slots of ϵW, we will show in next chapter that all the elements can be packed, and then we take a total size of at most $3\epsilon W$ of *small* elements outside overflowed trains and use additional trains to pack them. We will use an algorithm(algorithm 1) to allocate these elements, which is described in next chapter, we also prove that the algorithm assures all these customers are packed and the additional cost is at most $4\epsilon \cdot c(\mathcal{R}) + \mathbb{P}_1$, where $c(\mathcal{R})$ is the cost of path $\mathcal{R}$ and $\mathbb{P}_1$ is the largest position of all customers.

As the time complexity of our scheme, we have the following lemma, its proof is in the full version.

Lemma 6. *The time complexity of our scheme is* $n^{O(\epsilon^{-4})}$.

4 The Analysis of Correctness

We then analyze the layered graph, and show how to transfer an optimal *small expanse* solution into a corresponding path in the graph. In a path in the graph from the start vertex s to the terminal vertex t we pack *large* elements there, and keep space in the form of slots of size ϵW for *small* elements. We then packs *small* elements back into the path using these slots as well as one additional slot

we can make sure all the *small* elements can be packed into these space, and the content of a train may overflow to at most $W + 2\epsilon W$. We then take *small* elements with a total size at most $3\epsilon W$ from the overflowed train, and pack them into additional trains, finally analyze the cost of additional trains and give the bound of our solution.

4.1 Transfer an Optimal *Small Expanse* Solution to a Corresponding Path in the Graph

We first bound the cost of path $\mathcal{R}$. To do so, we present a path from s to t in the graph G whose cost is $(1 + 2\epsilon)OPT_n$, where OPT_n is an optimal *small expanse* solution. The proof is left in the full version.

Lemma 7. *There exists a path $\tilde{\mathcal{R}}$ from s to t whose cost $c(\tilde{\mathcal{R}})$ is at most* $(1 + 2\epsilon)OPT_n$.

4.2 Packing Backing *Small* Elements in a Path

Consider a path $\mathcal{R}$. Focus on stage i, let μ_i be the number of slots for *small* elements in stage i as implied by the sum of values $\hat{n}_{slot}$ in patterns(a pattern is equal to an edge of type one), therefore the space $\epsilon W \cdot \mu_i$ will be larger than the actual total size of *small* elements. Let M_i be the number of edges of type one inside stage that contain at least one slot for *small* elements, and N_i be the total number of edges of type one of stage i(total number of trains of this stage used by the path). We have a total size of $\epsilon W \cdot \mu_i$ space for packing back the *small* customers of stage i. For each edge that contains at least one slot for *small* elements, we give one additional slot ϵW for it, then there are $\mu_i + M_i$ slots for packing *small* elements whose total size is at most $\epsilon W \cdot \mu_i$. We pack the real *small* customers greedily into the space provided by these slots($\mu_i + M_i$ slots), if we have customers left and there no space available for them, at least a total size of $\epsilon W(\mu_i + M_i) - \epsilon W M_i$ of customers are already in the trains, which contradicted with the fact that the total size of all *small* customers is at most $\epsilon W \cdot \mu_i$. Therefore these $\mu_i + M_i$ slots are enough to allocate all the *small* customers. However some trains may overflow to $W + 2\epsilon W$. Next we show how to transform it into a feasible solution.

Let C be the content of a overflowed train as described above, we take the *small* customer out of C one at time, we keep doing this until we take one *small* element s_i out of C and the content of C becomes at most W. This operation will take out *small* customers with total size at most $3\epsilon W$, since if not the total size of elements before taking out s_i is at most $2\epsilon W$ and s_i is larger than ϵW, which makes a contradiction. We then pack all these customers taken from overflowed trains with new trains by algorithm 1, which makes sure all these customers are packed and the additional cost is at most $4\epsilon \cdot c(\mathcal{R}) + \mathbb{P}_1$, where $c(\mathcal{R})$ is the cost path $\mathcal{R}$, $\mathbb{P}_1$ is the largest position of all customers.

Let $\mathbb{C}_i$ be the set of *small* customers that are taken from the overflowed trains of stage i as described above. Let list L_c be formed of the *small* customers of $\mathbb{C}_i$ followed by *small* customers of $\mathbb{C}_{i+1}$ for $1 \leq i \leq r$. We will pack the elements of L_c by algorithm 1.

Algorithm 1. packing overflowed *small* customers with additional trains

```
INPUT: $L_c$.
    $k = 0$.
    while $L_c \neq \emptyset$
      $k = k + 1$.
      Assume the first element in $L_c$ is of position $\mathbb{P}_j$.
      Open a new train of cost $\mathbb{P}_j$.
      $Cost[k] = \mathbb{P}_j$.
      while the first element of $L_c$ can be put into this train
        Take out the first element from list $L_c$ and put it into the train.
      end while
    end while
OUTPUT: $k$ trains packing $L_c$.
```

Lemma 8. *The total cost of additional trains for packing small elements is at most* $4\epsilon \cdot c(\mathcal{R}) + \mathbb{P}_1$ *(the proof is left in the full version).*

By Lemmas 6, 7, and 8, we have the following result:

Theorem 1. *The scheme described above is an APTAS for TDP.*

5 Conclusion

In order to get an APTAS for the train delivery problem, we first applying rounding techniques on positions and sizes by Lemmas 1 and 3. We partition input elements into sets by their positions which we called classes. We apply linear grouping on these classes so that each class has a constant number of different positions, then we construct layered graph on these adapted input elements and find a feasible solution which we prove is a near optimal solution to the original input.

Acknowledgements. This research has been partially supported by "the Fundamental Research Funds for the Central Universities(DUT12LK09)" and NSFC (11101065), RGC(HKU716412E).

References

1. Coffman, E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: Hochbaum, D. (ed.) Approximation algorithms. PWS Publishing Company (1997)
2. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica (1981)
3. Das, A., Mathieu, C., Mozes, S.: The Train Delivery Problem - Vehicle Routing Meets Bin Packing. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 94–105. Springer, Heidelberg (2011)
4. Murgolo, F.D.: An efficient approximation scheme for variable-sized bin packing. SIAM J. Comput. 16(1), 149–161 (1987)
5. Garey, M.R., Johnson, D.S.: "strong" np-completeness results: Motivation, examples, and implications. J. ACM 25(3), 499–508 (1978)
6. Friesen, D.K., Langston, M.A.: Variable sized bin packing. SIAM J. Comput. 15(1), 222–230 (1986)
7. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: FOCS, pp. 312–320. IEEE Computer Society, Washington, DC (1982)
8. Kang, J., Park, S.: Algorithm for the variable sized bin packing problem. European Journal of Operational Research 147(2), 365–372 (2003)
9. Seiden, S.S., van Stee, R., Epstein, L.: New bounds for variable-sized online bin packing. SIAM Journal on Computing 32(2), 455–469 (2003)
10. Epstein, L., Levin, A.: An APTAS for Generalized Cost Variable-Sized Bin Packing. SIAM J. Comput. 38(1), 411–428 (2008)
11. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processor: Using the dual approximation approach. SIAM Journal on Computing 17, 539–551 (1988)
12. Haimovich, M., Rinnooy Kan, A.H.G., Stougie, L.: Analysis of heuristics for vehicle routing problems. In: Vehicle Routing: Methods and Studies. Management Sci. System, vol. 16, pp. 47–61. North Holland, Amsterdam, Elsevier Scinece B.V. This is a full in book entry (1988)

The Distance 4-Sector of Two Points Is Unique

Robert Fraser[1], Meng He[2], Akitoshi Kawamura[3], Alejandro López-Ortiz[4], J. Ian Munro[4], and Patrick K. Nicholson[5]

[1] Department of Computer Science, University of Manitoba, Winnipeg, Canada
`fraser@cs.umanitoba.ca`

[2] Faculty of Computer Science, Dalhousie University, Halifax, Canada
`mhe@cs.dal.ca`

[3] Department of Computer Science, University of Tokyo, Tokyo, Japan
`kawamura@is.s.u-tokyo.ac.jp`

[4] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
`{alopez-o,imunro}@cs.uwaterloo.ca`

[5] Max-Planck-Institut für Informatik, Saarbrücken, Germany
`pnichols@mpi-inf.mpg.de`

Abstract. The (distance) k-sector is a generalization of the concept of bisectors proposed by Asano, Matoušek and Tokuyama. We prove the uniqueness of the 4-sector of two points in the Euclidean plane. Despite the simplicity of the unique 4-sector (which consists of a line and two parabolas), our proof is quite non-trivial. We begin by establishing uniqueness in a small region of the plane, which we show may be perpetually expanded afterward.

Keywords: distance k-sector, Tarski fixed point, uniqueness.

1 Introduction

The *bisector* of two nonempty sets X and Y in $\mathbb{R}^2$ is defined as

$$\operatorname{bisect}(X,Y) = \{\, z \in \mathbb{R}^2 : \operatorname{dist}(z,X) = \operatorname{dist}(z,Y) \,\}, \tag{1}$$

where $\operatorname{dist}(z,X) = \inf_{x\in X} \operatorname{dist}(z,x)$ denotes the Euclidean distance of z from a set X. For any integer $k \geq 2$, a *distance k-sector* (or simply *k-sector*) of distinct points p, $q \in \mathbb{R}^2$ is a $(k-1)$-tuple $(C_1,\ldots,C_{k-1})$ of nonempty subsets of $\mathbb{R}^2$ such that

$$C_i = \operatorname{bisect}(C_{i-1}, C_{i+1}), \qquad i = 1,\ldots,k-1, \tag{2}$$

where $C_0 = \{p\}$ and $C_k = \{q\}$.

For example, there is a 4-sector of two points that consists of a line and two parabolas (Fig. 1). We will prove that this is the only one:

Theorem 1. *The 4-sector between two points in the Euclidean plane is unique.*

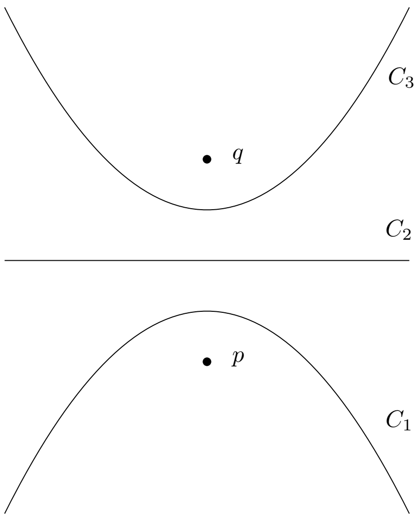


Fig. 1. A 4-sector (C_1, C_2, C_3) of p and q (whose uniqueness we will prove). If $p = (0, -1)$ and $q = (0, 1)$, these curves are the graphs of $y = \pm(x^2 + 1)/2$ and $y = 0$.

The notion of distance k-sectors was introduced by Asano and Tokuyama [4] in 2004, motivated by a question about circuit board design. Asano, Matoušek and Tokuyama [3] showed the existence and uniqueness of the 3-sector (trisector). Despite the simple definition, k-sectors (with the exception of $k = 2, 4$) do not seem to be easy to construct (note that bisecting between the curves in Fig. 1 does *not* give an 8-sector). In particular, the 3-sector is not algebraic, as conjectured in [3] and recently proved by Monterde and Ongay [8].

Although the original proof in [3] of the existence and uniqueness of the 3-sector was rather involved, it turned out later that k-sectors exist in a fairly general setting and for a relatively simple reason: Reem and Reich [9] used the Tarski fixed point theorem to prove the existence of a closely related object called *double zone diagrams.* Applying this idea, Imai et al. [6] proved the existence of a k-sector for any k on any sets P, Q (instead of $\{p\}$, $\{q\}$), and for a general class of metric spaces. The existence proofs sometimes suggest an algorithm to compute (draw on a screen approximately) k-sectors, but its efficiency is unclear. Some issues in computing k-sectors and double zone diagrams are discussed in [6,10] (of course, such issues are trivial in our setting of 4-secting two points, once we show that the only 4-sector is the one explicitly given in Fig. 1).

Uniqueness is harder to prove. The uniqueness of the 3-sector was extended to the case where one of P, Q is a line segment instead of a point [5], and then to the general case where P, Q can be any disjoint nonempty closed sets [7] (in fact, they proved the uniqueness of the *zone diagram* [2], a generalization of 3-sectors where we start with many sets instead of just two sets P, Q). Unlike existence, uniqueness relies on the properties of the Euclidean norm, and indeed fails for, say, the l_1 norm [1,7]. It remains open [6, Conjecture] whether k-sectors, for $k \geq 4$, are unique even in the Euclidean plane. Theorem 1 answers this for $k = 4$ (and points p, q).

Proof idea. Roughly speaking, the central part of our proof (Lemma 3) is based on the following ideas. Suppose that there are two different k-sectors $(C_i)_i$ and $(\hat{C}_i)_i$ (here and in the sequel, the subscript i always ranges over $1, \ldots, k-1$, so that $(C_i)_i$ means $(C_1, \ldots, C_{k-1})$). Then there is a gap somewhere between the curves C_i and $\hat{C}_i$. Since they both satisfy equation (2), we must have another gap somewhere between C_{i-1} and $\hat{C}_{i-1}$ or between C_{i+1} and $\hat{C}_{i+1}$, which is not too small compared to the original gap. From some observations about the size and location of the new gap, we derive contradiction by arguing that this process of finding a new gap cannot go on forever because it causes the gap to grow too big to fit where it must be. This rough intuition is common to the proof of 3-sector uniqueness in [7], but there is a lot of room for creativity as to how we define the size of the gap between two curves at a point. The proof in [7] used a clever way to measure the gap under which the gap always grows bigger, but this measure of the gaps only makes sense for 3-sectors. We measure the gap much more simply, by the difference between the y-coordinates of the two curves at a common x-coordinate. The downside is that under this measure, the gap gets bigger only when the involved parts of the curves lie in certain configuration. This necessitates some detailed argument that certain part of 4-sector indeed has this configuration (Lemma 6) and that the uniqueness of this part of the 4-sector can then be extended gradually to other parts (Lemmas 4 and 5).

2 Preliminaries: Gradations

In the definition of k-sectors above, the components C_i are sets satisfying certain equations, and we did not even say that they are curves. It is not entirely obvious, although it is true, that each of them divides the plane into two regions, one containing p and the other containing q. Imai et al. [6] made this claim precise (as Lemma 1 below) by introducing *k-gradations*. We briefly review their definition, as we also find k-gradations easier to reason with than k-sectors.

For nonempty X, $Y \subseteq \mathbb{R}^2$, we define the *dominance region* of X over Y by

$$\operatorname{dom}(X, Y) = \{\, z \in \mathbb{R}^2 : \operatorname{dist}(z, X) \leq \operatorname{dist}(z, Y) \,\}. \tag{3}$$

It is not hard to see [6, Lemma 6] that, if X and Y are disjoint closed sets, $\operatorname{bisect}(X, Y)$ is the boundary of $\operatorname{dom}(X, Y)$. A *k-gradation* between points p, $q \in \mathbb{R}^2$ is a $(k-1)$-tuple $(R_i, S_i)_i$ of pairs of subsets of $\mathbb{R}^2$ satisfying

$$R_i = \operatorname{dom}(R_{i-1}, S_{i+1}), \qquad S_i = \operatorname{dom}(S_{i+1}, R_{i-1}), \qquad i = 1, \ldots, k-1, \tag{4}$$

where $R_0 = \{p\}$ and $S_k = \{q\}$. It is easy to see that this implies $R_i \cup S_i = \mathbb{R}^2$, $\{p\} = R_0 \subseteq R_1 \subseteq \cdots \subseteq R_{k-1}$ and $S_1 \supseteq S_2 \supseteq \cdots \supseteq S_k = \{q\}$.

By the following lemma (a special case of [6, Proposition 2]), a k-sector $(C_i)_i$ can be identified with a k-gradation $(R_i, S_i)_i$:

Lemma 1 ([6]). *A $(k-1)$-tuple $(C_i)_i$ of sets is a k-sector of points p, $q \in \mathbb{R}^2$ if and only if*

$$C_i = R_i \cap S_i, \qquad i = 1, \dots, k-1 \tag{5}$$

for some k-gradation $(R_i, S_i)_i$ between p, q.

Imai et al. [6, Proposition 1] established the existence of a k-sector by proving the existence of a k-gradation:

Lemma 2 ([6]). *There exists a k-gradation between two distinct points in $\mathbb{R}^2$. In fact, there are the greatest and the least k-gradations under the order defined by: $(R_i, S_i)_i \le (R'_i, S'_i)_i$ if $R_i \subseteq R'_i$ and $S_i \supseteq S'_i$ for all $i = 1, \dots, k-1$.*

In what follows, we will primarily deal with k-gradations as opposed to k-sectors, as their use allows the proofs to be simpler and cleaner.

3 Proof of Theorem 1

For the rest of the paper, we fix $p = (0, -1)$ and $q = (0, +1)$ and consider the 4-sectors and 4-gradations between them. As explained above (Lemma 1), each 4-sector $(C_i)_i$ corresponds to a 4-gradation $(R_i, S_i)_i$. We will prove that this 4-gradation agrees with $(\hat{R}_i, \hat{S}_i)_i$, the trivial 4-gradation corresponding to the trivial 4-sector $(\hat{C}_i)_i$ described in Fig. 1.

Define the (closed) regions I_β and J_β, for $\beta > 0$, as follows (Fig. 2), using the trivial 4-sector $(\hat{C}_i)_i$. Consider the normal lines to the parabolas $\hat{C}_1$ and $\hat{C}_3$ at $x = \pm\beta$. The closed, finite region of the plane (containing the origin) defined by these lines is I_β. The region J_β contains I_β, as well as all points lying above the upper envelope of the normal lines to $\hat{C}_1$ and all points lying below the lower envelope of the normal lines to $\hat{C}_3$. Calculation shows that the four vertices of I_β are $(\pm(3+\beta^2)\beta/2, 0)$ and $(0, \pm(3+\beta^2)/2)$.

We prove Theorem 1 using the following three lemmas. Note that the number 0.5774 in the lemmas is slightly greater than $1/\sqrt{3}$, so that the boundary halflines of $J_{0.5774}$ make angle slightly less than $\pi/3$ with the x-axis.

Lemma 3. *Let $(R_i, S_i)_i$ be a 4-gradation. Then R_2 and S_2 agree with the trivial 4-gradation on $J' := J_{0.5774} \cap (\mathbb{R} \times [-4, 4])$, i.e., $R_2 \cap J' = \hat{R}_2 \cap J'$ and $S_2 \cap J' = \hat{S}_2 \cap J'$.*

Lemma 4. *Let $\beta \ge 0.5774$. Let $(R_i, S_i)_i$ be a 4-gradation. Suppose that R_2 and S_2 agree with the trivial 4-gradation on J_β, i.e., $R_2 \cap J_\beta = \hat{R}_2 \cap J_\beta$ and $S_2 \cap J_\beta = \hat{S}_2 \cap J_\beta$. Then R_1, S_1, R_3, S_3 agree with the trivial 4-gradation on a region $I_{\beta'}$, for some $\beta' \ge 1.00001\beta$. Moreover, for $\beta = 0.5774$, this holds with J_β replaced by the J' defined in Lemma 3.*

Lemma 5. *Let $\beta > 0$. Let $(R_i, S_i)_i$ be a 4-gradation. Suppose that R_1, S_1, R_3, S_3 agree with the trivial 4-gradation on I_β, i.e., $R_i \cap I_\beta = \hat{R}_i \cap I_\beta$ and $S_i \cap I_\beta = \hat{S}_i \cap I_\beta$ for each $i = 1, 3$. Then R_2, S_2 agree with the trivial 4-gradation on J_β, i.e., $R_2 \cap J_\beta = \hat{R}_2 \cap J_\beta$ and $S_2 \cap J_\beta = \hat{S}_2 \cap J_\beta$.*

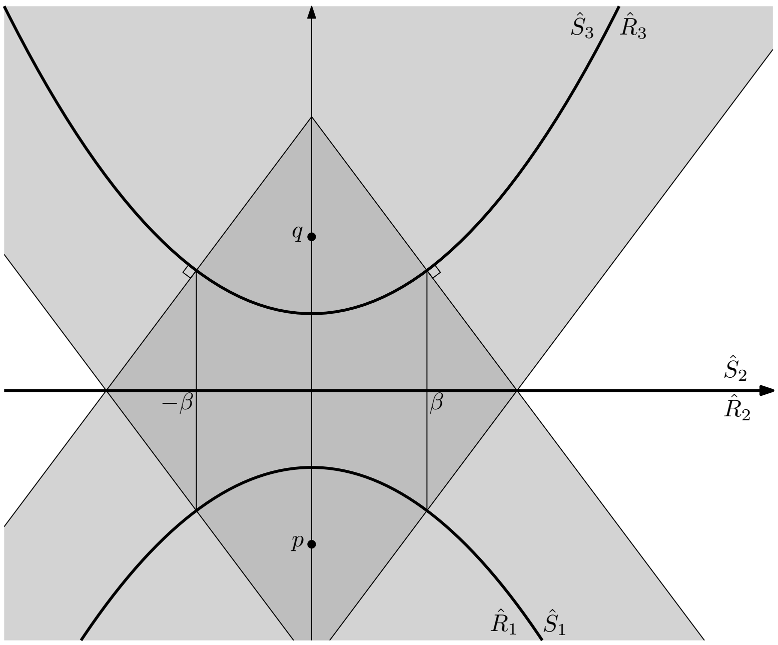


Fig. 2. Closed regions I_β (dark grey) and J_β (light grey plus dark grey), defined from the trivial 4-gradation $(\hat{R}_i, \hat{S}_i)_i$

Lemma 3 is the central part of our proof: it says that the middle component (R_2, S_2) of the 4-gradation is uniquely determined (and agrees with the trivial one) up to a certain point (i.e., in the region J'). Lemma 4 says that if (R_2, S_2) is uniquely determined up to some point, then it uniquely determines (R_1, S_1) and (R_3, S_3) up to some point. Lemma 5 works the other way, stating that a certain part of (R_2, S_2) is determined by certain parts of (R_1, S_1) and (R_3, S_3).

To prove Theorem 1, we use Lemma 3 followed by the last sentence (the "moreover" part) of Lemma 4, and then apply Lemmas 5 and 4 alternately, extending step by step the region on which the considered 4-gradation agrees with the trivial one. This proves the uniqueness of the 4-gradation, and thus, by Lemma 1, of the 4-sector.

The proofs of the lemmas require a number of additional properties, and so their presentation is delayed to Sections 3.2 and 3.3.

3.1 Properties of a 4-Gradation

Before proving the lemmas, we need to get some rough estimates about what a 4-gradation must look like. Let $\mathrm{B}(w, r)$ (resp. $\mathrm{B}^\circ(w, r)$) denote the closed (resp. open) ball with centre w and radius r.

Lemma 6. *Let $(R_i, S_i)_i$ be a 4-gradation. Then*

1. *R_1 and S_3 are convex.*
2. $\mathrm{dist}(R_i, S_{i+1}) \geq 1/2$ *for each* $i = 0, 1, 2, 3$.
3. $\mathrm{B}(p, i/2) \subseteq R_i$ *and* $\mathrm{B}(q, i/2) \subseteq S_{4-i}$ *for each* $i = 0, 1, 2, 3, 4$.
4. $R_1 \subseteq \mathbb{R} \times (-\infty, -1/2]$ *and* $S_3 \subseteq \mathbb{R} \times [1/2, \infty)$.
5. *Let $w = (a, b)$ be a point with $a \geq 0$ and $b \leq 0$. Let $v = (a', b')$ be a closest point to w in S_3. Then,*
   (a) $0 \leq a' \leq a$.
   (b) If $|a| \leq 1.16$*, then* $b' \in [1/2, 0.82]$.
   Likewise for S_3 and $b \leq 0$ replaced by R_1 and $b \geq 0$, respectively.
6. *For each $t > 0$, let $L_t^{\pm} = \{ (x, y) \in \mathbb{R}^2 : \pm y \leq x^2/t \}$, respectively. Then $R_2 \subseteq L_4^+$ and $S_2 \subseteq L_4^-$.*

Proof. 1. $R_1 = \mathrm{dom}(p, S_2) = \bigcap_{s \in S_2} \mathrm{dom}(p, s)$, and the right-hand side is an intersection of halfspaces. Similarly for S_3.

2. Otherwise, there exists an i such that $\mathrm{dist}(x_i, x_{i+1}) < 1/2$ for some $x_i \in R_i$ and $x_{i+1} \in S_{i+1}$. By redefining x_i and x_{i+1} on the segment $x_i x_{i+1}$ if necessary, we have $x_i \in R_i \cap S_i$ and $x_{i+1} \in R_{i+1} \cap S_{i+1}$. Starting from this, we can inductively obtain a point $x_j \in R_j \cap S_j$ with $\mathrm{dist}(x_j, x_{j+1}) < 1/2$ for each $j = i-1, i-2, \ldots, 0$; and a point $x_j \in R_j \cap S_j$ with $\mathrm{dist}(x_{j-1}, x_j) < 1/2$ for each $j = i+2, i+3, \ldots, 4$. Thus p and q are connected by a path $x_0 x_1 x_2 x_3 x_4$ of length less than $4 \times 1/2$, a contradiction.
3. We prove the first claim by induction on i (the second claim is analogous). By part 2 and the induction hypothesis, $S_{i+1} \subseteq \mathbb{R}^2 \setminus \mathrm{B}^\circ(p, (i+1)/2)$. By this and $R_{i+1} \cup S_{i+1} = \mathbb{R}^2$, we have $R_{i+1} \supseteq \mathrm{B}^\circ(p, (i+1)/2)$. Since R_{i+1} is closed, $R_{i+1} \supseteq \mathrm{B}(p, (i+1)/2)$.
4. Since $(0, 0) \in S_2$ by part 3, we have $R_1 = \mathrm{dom}(\{p\}, S_2) \subseteq \mathrm{dom}(\{p\}, (0, 0)) = \mathbb{R} \times (-\infty, -1/2]$. Similarly for S_3.
5. Since S_3 is convex by part 1, it is contained in the (closed) halfplane H_w defined as the opposite side from w across the perpendicular of vw at v. Since H_w contains $\mathrm{B}(q, 1/2)$ by part 3, and the point v on its boundary is in $\mathbb{R} \times [1/2, \infty)$ by part 4, it must be the case that $0 \leq a' \leq a$.
   We will prove the other claim, $b' \leq 0.82$. It suffices to show this for the case where w is the upper-right-most point $w_0 = (1.16, 0)$. To see why, let $v_0 = (a'_0, b'_0)$ be a closest point in S_3 to w_0, and define the halfplane H_{w_0} in the same way as H_w from w_0 and v_0. Since H_{w_0} contains w and H_w contains w_0, they must coincide. So v must be at the same relative position from v_0 as w is from w_0, and thus must lie below, i.e., $b' \leq b'_0$.
   Thus our goal now is to show $b'_0 \leq 0.82$. Recall that H_{w_0} contains $\mathrm{B}(q, 1/2)$. The line tangent to the lower half of the ball $\mathrm{B}(q, 1/2)$ at the point with x-coordinate x_0 is described as

$$y = \frac{x_0}{\sqrt{1/4 - x_0^2}} x + \frac{\sqrt{1/4 - x_0^2} - 1/4}{\sqrt{1/4 - x_0^2}} \quad . \tag{6}$$

   Similarly, the line perpendicular to this tangent and passing through w_0 is

$$y = -\frac{\sqrt{1/4 - x_0^2}}{x_0} x + \frac{1.16\sqrt{1/4 - x_0^2}}{x_0} \quad . \tag{7}$$

These lines intersect at

$$x = \frac{1.16/4 + x_0/4 - 1.16x_0^2 - x_0\sqrt{1/4 - x_0^2}}{1/4 + x_0^2} . \tag{8}$$

Using substitution and a numerical calculation we get that the maximum y-value achieved by this intersection point over the domain $x_0 \in (0, 1/2)$ is $0.819534\ldots < 0.82$ at $x_0 = 0.243934\ldots$. This value bounds b_0' from above.

6. By parts 3 and 4, $R_2 = \mathrm{dom}(R_1, S_3) \subseteq \mathrm{dom}(\mathbb{R} \times (-\infty, -1/2], \mathrm{B}(q, 1/2)) = \mathrm{dom}(\mathbb{R} \times (-\infty, -1]), \{q\}) = L_4^+$. Similarly for S_2. □

In the rest of this paper, part x of Lemma 6 is referred to as Lemma 6.x.

3.2 Proof of Lemma 3: Uniqueness Near the Origin

We now prove Lemma 3. Recall the statement:

Lemma 3. Let $(R_i, S_i)_i$ be a 4-gradation. Then R_2 and S_2 agree with the trivial 4-gradation on $J' := J_{0.5774} \cap (\mathbb{R} \times [-4, 4])$, i.e., $R_2 \cap J' = \hat{R}_2 \cap J'$ and $S_2 \cap J' = \hat{S}_2 \cap J'$.

Proof. We may assume that $(R_i, S_i)_i$ is the greatest 4-gradation in the sense of Lemma 2 (the same argument works for the least gradation by symmetry, and proves that the greatest and least gradations, and hence all gradations, coincide on J'). This implies that $R_i \supseteq \hat{R}_i$ and $S_i \subseteq \hat{S}_i$ for each i. It suffices to prove that $R_2 \cap J' \subseteq \hat{R}_2 \cap J'$. Suppose otherwise, i.e., that $R_2 \cap J'$ contains a point above the x-axis. Then the set of $t > 0$ with $L_t^+ \supseteq R_2 \cap J'$ (see Lemma 6.6 for the definition of L_t^+) is bounded from above. This set is closed, and contains 4 by Lemma 6.6, so it has a maximum, which we call t from now on. By the maximality of t, there is a point $w = (a, \varepsilon)$ in $(R_2 \cap J') \setminus \{(0, 0)\}$ on the boundary of L_t^+. This point w is on the boundary of R_2. We assume without loss of generality that $a > 0$. Calculation shows that the right uppermost point of $J' \cap L_4^+$ is $(1.15487\ldots, 0.33343\ldots)$, so $a < 1.16$ and $\varepsilon < 0.34$. Let $\hat{w} := (a, 0)$.

Thus, we are looking at the point $w \in R_2 \cap J'$ that lies on the boundary of L_t^+. Below, we will split into two cases, and argue that in either case we can find another point $u \in R_2 \cap J'$ that does not belong to L_t^+, contradicting the way we defined t. Let $l := \mathrm{dist}(w, R_1) = \mathrm{dist}(w, S_3)$ and $\hat{l} := \mathrm{dist}(\hat{w}, \hat{R}_1) = \mathrm{dist}(\hat{w}, \hat{S}_3)$.

Case 1: $l \leq \hat{l}$. Define v to be the closest point to w in R_1, and let $\hat{v} := v - (0, \varepsilon)$ (Fig. 3). Since $\mathrm{dist}(\hat{w}, \hat{v}) = \mathrm{dist}(w, v) = l$, we have $\hat{v} \in \mathrm{B}(\hat{w}, l) \subseteq \mathrm{B}(\hat{w}, \hat{l}) \subseteq \hat{S}_1$.

Let $h := \mathrm{dist}(v, S_2)$ and $\hat{h} := \mathrm{dist}(\hat{v}, \hat{S}_2)$. Since $v \in \mathbb{R} \times [-0.82, 0.82]$ by Lemma 6.5b, and $\varepsilon < 0.34$, we have $h = \mathrm{dist}(v, p) > \mathrm{dist}(\hat{v}, p) \geq \hat{h}$.

Define $\hat{u}$ to be the closest point to $\hat{v}$ in $\hat{S}_2$ and let $u := \hat{u} + (0, \varepsilon) = (a', \varepsilon)$. Thus, u and $\hat{u}$ are vertically aligned with $\hat{v}$. Since $\mathrm{dist}(v, u) = \mathrm{dist}(\hat{v}, \hat{u}) = \hat{h} < h$, the point u belongs to R_2 and is different from w (because $\mathrm{dist}(v, w) \geq h$). By Lemma 6.5a, it lies on the left of w, which was in J' and on the boundary of L_t^+. Hence, $u \in J' \setminus L_t^+$, contradicting $R_2 \cap J' \subseteq L_t^+$.

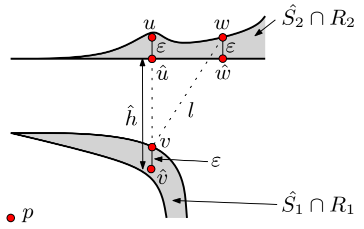


Fig. 3. Illustration of Case 1 from the proof of Lemma 3

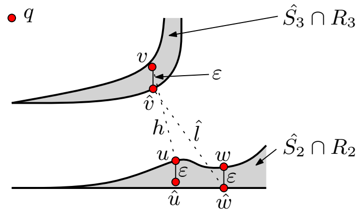


Fig. 4. Illustration of Case 2 from the proof of Lemma 3

Case 2: $l \geq \hat{l}$. Define $\hat{v}$ to be the closest point to $\hat{w}$ in $\hat{S}_3$, and let $v := \hat{v} + (0, \varepsilon)$ (Fig. 4). Since $\operatorname{dist}(w, v) = \operatorname{dist}(\hat{w}, \hat{v}) = \hat{l}$, we have $v \in \mathrm{B}(w, \hat{l}) \subseteq \mathrm{B}(w, l) \subseteq R_3$.

Let $h := \operatorname{dist}(v, R_2)$ and $\hat{h} := \operatorname{dist}(\hat{v}, \hat{R}_2)$. Since $\hat{v} \in \mathbb{R} \times [-0.82, 0.82]$ by Lemma 6.5b, and $\varepsilon < 0.34$, we have $h \leq \operatorname{dist}(v, q) < \operatorname{dist}(\hat{v}, q) = \hat{h}$.

Define u to be the closest point (or one of the closest points) to v in R_2, and let $\hat{u} = u - (0, \varepsilon)$. Since $w \in J' \cap R_2 \subseteq J' \cap L_4^+$ by Lemma 6.6, the point v is in the region V in Fig. 5. Calculation shows that V does not intersect $\operatorname{dom}(\mathbb{R}^2 \setminus J', \{q\})$. In particular, $\operatorname{dist}(v, \mathbb{R}^2 \setminus J') > \operatorname{dist}(v, q) \geq h = \operatorname{dist}(v, u)$. Hence $u \in J'$.

Again, our goal is to prove $u \notin L_t^+$, thus contradicting $R_2 \cap J' \subseteq L_t^+$. Note that $u \in \mathrm{B}(v, \hat{l})$, because u is a closest point to v in R_2, and hence closer than w. Note also that $u \in \mathbb{R} \times (\varepsilon, \infty)$, because $\hat{u}$ lies in the interior of $\mathrm{B}(\hat{v}, \hat{h}) \subseteq \hat{S}_2$ by $\operatorname{dist}(\hat{v}, \hat{u}) = \operatorname{dist}(v, u) = h < \hat{h}$. Thus, it suffices to prove that the three regions L_t^+, $\mathrm{B}(v, \hat{l})$ and $\mathbb{R} \times (\varepsilon, \infty)$ do not intersect. We argue as follows.

Notice that the boundaries of these three regions all pass through w. At this point w, the boundary of $\mathrm{B}(v, \hat{l})$ has slope b, where b is the x-coordinate of $\hat{v}$ and v, and the boundary of L_t^+ has slope $2a/t$, which is smaller than b because $a = (3 + b^2)b/2 < 2b$ and $t \geq 4$. And also on the right of w, the slope of the boundary of $\mathrm{B}(v, \hat{l})$ continues to be greater than that of L_t^+, because the curvature of the former is greater than that of the latter by $\hat{l} \leq \operatorname{dist}((1.16, 0), \hat{S}_3) = 0.874\ldots < 1$. Hence, the boundaries of $\mathrm{B}(v, \hat{l})$ and L_t^+ never meet on the right of w, so the intersection $\mathrm{B}(v, \hat{l}) \cap L_t^+$ lies entirely below w, and thus misses $\mathbb{R} \times (\varepsilon, \infty)$. □

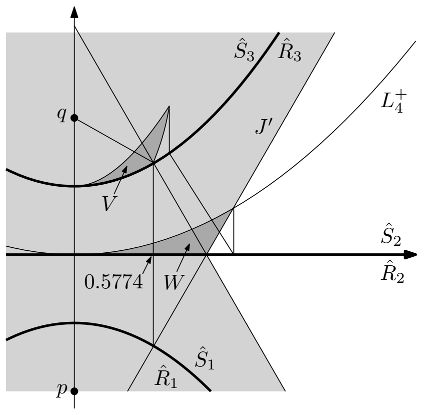


Fig. 5. The light-shaded region is J'. Since the point w is in the region $W = L_4^+ \cap \hat{S}_2 \cap J'$, the point v (obtained from w by way of $\hat{w}$ and $\hat{v}$ as described in Case 2 of the proof of Lemma 3) is in the region V (defined by suitable algebraic curves).

3.3 Proof of Lemmas 4 and 5: Extending Uniqueness

We define an additional closed region K_β. The boundaries of K_β are line segments which are positioned at angles of $\pm\pi/3$ with respect to the x-axis and intersect the boundary of I_β on the x-axis, so the vertices bounding K_β are $(\pm(3+\beta^2)\beta/2, 0)$ and $(0, \pm(3+\beta^2)\sqrt{3}\,\beta/2)$. Note $I_\beta \subset K_\beta$ when $\beta > 1/\sqrt{3}$.

Recall Lemma 4:

Lemma 4. Let $\beta \geq 0.5774$. Let $(R_i, S_i)_i$ be a 4-gradation. Suppose that R_2 and S_2 agree with the trivial 4-gradation on J_β, i.e., $R_2 \cap J_\beta = \hat{R}_2 \cap J_\beta$ and $S_2 \cap J_\beta = \hat{S}_2 \cap J_\beta$. Then R_1, S_1, R_3, S_3 agree with the trivial 4-gradation on a region $I_{\beta'}$, for some $\beta' \geq 1.00001\beta$. Moreover, for $\beta = 0.5774$, this holds with J_β replaced by the J' defined in Lemma 3.

Proof. Let $z \in K_\beta$. We demonstrate that $K_\beta \cap R_1 = K_\beta \cap \hat{R}_1$; the argument is similar for S_1, R_3, S_3. The condition $\beta \geq 0.5774 > 1/\sqrt{3}$ ensures that the lines delimiting J_β have slope smaller (in absolute value) than $\sqrt{3}$, so that any point in K_β is closer to $\hat{S}_2 \cap J_\beta$ (which coincides with $S_2 \cap J_\beta$) than to $\mathbb{R}^2 \setminus J_\beta$. So the nearest point to z in S_2 is in J_β. Hence

$$\mathrm{dist}(z, S_2) = \mathrm{dist}(z, S_2 \cap J_\beta) = \mathrm{dist}(z, \hat{S}_2 \cap J_\beta) = \mathrm{dist}(z, \hat{S}_2). \qquad (9)$$

So $z \in R_1$ if and only if $z \in \hat{R}_1$. Since z was an arbitrary element of K_β, we have $K_\beta \cap R_1 = K_\beta \cap \hat{R}_1$.

Define β' to be the absolute value of the x-coordinate of an intersection point of R_1 (or equivalently $\hat{R}_1$) with the boundary of K_β. Since R_1 is convex, it cannot contain a point in $I_{\beta'} \setminus K_\beta$, and thus $I_{\beta'} \cap R_1 = I_{\beta'} \cap \hat{R}_1$.

It remains to prove $\beta' \geq 1.00001\beta$. To express β' in terms of β, it suffices to equate the expressions for C_1 and the boundary of K_β and solve for the roots:

$$\beta' = -\sqrt{3} + \sqrt{2 + \sqrt{3}\beta^3 + 3\sqrt{3}\beta}. \tag{10}$$

Finally, it may be shown that $\sigma' \geq 1.00001\sigma$ for $\sigma' := \sqrt{3}\,\beta'$ and $\sigma := \sqrt{3}\,\beta \geq 0.5774\sqrt{3} \geq 1.00001^8$. Calculation shows that

$$\sigma' = -3 + \sqrt{6 + 9\sigma + \sigma^3} \geq -3 + (3 + \sigma^{9/8}) = \sigma^{1/8} \cdot \sigma \geq 1.00001\sigma, \tag{11}$$

where the first inequality holds because $(6 + 9\sigma + \sigma^3) - (3 + \sigma^{9/8})^2 = 3(\sigma - 1) + \sigma(\sigma^{10/8} + \sigma^{11/8} + \sigma^{12/8} + \sigma^{13/8} + \sigma^{14/8} + \sigma^{15/8} - 6)(\sigma^{1/8} - 1) \geq 0$.

For the final statement of the lemma (the "moreover" part), note that if $\beta = 0.5774$, then $(3 + \beta^2)\sqrt{3}\,\beta/2 < 1.66$. Thus, $K_{0.5774} \subset \mathbb{R} \times [-1.66, 1.66]$, and so points in $J_{0.5774} \setminus J'$ are irrelevant. □

Finally, we prove Lemma 5. Recall its statement:

Lemma 5. Let $\beta > 0$. Let $(R_i, S_i)_i$ be a 4-gradation. Suppose that R_1, S_1, R_3, S_3 agree with the trivial 4-gradation on I_β, i.e., $R_i \cap I_\beta = \hat{R}_i \cap I_\beta$ and $S_i \cap I_\beta = \hat{S}_i \cap I_\beta$ for each $i = 1, 3$. Then R_2, S_2 agree with the trivial 4-gradation on J_β, i.e., $R_2 \cap J_\beta = \hat{R}_2 \cap J_\beta$ and $S_2 \cap J_\beta = \hat{S}_2 \cap J_\beta$.

Proof. Let $z \in J_\beta$. Because R_1 is convex (Lemma 6.1) and agrees with $\hat{R}_1$ on I_β, the nearest point to z in R_1 is in I_β. Hence

$$\operatorname{dist}(z, R_1) = \operatorname{dist}(z, R_1 \cap I_\beta) = \operatorname{dist}(z, \hat{R}_1 \cap I_\beta) = \operatorname{dist}(z, \hat{R}_1), \tag{12}$$

and likewise

$$\operatorname{dist}(z, S_3) = \operatorname{dist}(z, S_3 \cap I_\beta) = \operatorname{dist}(z, \hat{S}_3 \cap I_\beta) = \operatorname{dist}(z, \hat{S}_3). \tag{13}$$

So $z \in S_2$ if and only if $z \in \hat{S}_2$. Since z was an arbitrary element of J_β, we have $J_\beta \cap S_2 = J_\beta \cap \hat{S}_2$. Similarly for R_2. □

This completes the proof of Theorem 1.

4 Conclusions

We have shown that the 4-sector of two points is unique. Although we tried our best to simplify our proof, we still find it frustrating (and intriguing) that the uniqueness of such a seemingly basic object needed several pages to prove. As mentioned in the introduction, the main idea was to argue that if there are two k-sectors that differ at some point, there must be another point where they differ "more", in terms of some measure of the difference. But to implement this idea, we had to resort to calculation that relied heavily on the special setting of $k = 4$ and P, Q being points. It would be nice if this tedious calculation could be replaced by a conceptually simpler argument that works in more general settings.

References

1. Asano, T., Kirkpatrick, D.: Distance trisector curves in regular convex distance metrics. In: Proc. 3rd International Symposium on Voronoi Diagrams in Science and Engineering, pp. 8–17 (2006)
2. Asano, T., Matoušek, J., Tokuyama, T.: Zone diagrams: Existence, uniqueness, and algorithmic challenge. SIAM Journal on Computing 37(4), 1182–1198 (2007)
3. Asano, T., Matoušek, J., Tokuyama, T.: The distance trisector curve. Advances in Mathematics 212(1), 338–360 (2007)
4. Asano, T., Tokuyama, T.: Drawing equally-spaced curves between two points. In: Proc. 14th Fall Workshop on Computational Geometry, pp. 24–25 (2004)
5. Chun, J., Okada, Y., Tokuyama, T.: Distance trisector of a segment and a point. Interdisciplinary Information Sciences 16(1), 119–125 (2010)
6. Imai, K., Kawamura, A., Matoušek, J., Reem, D., Tokuyama, T.: Distance k-sectors exist. Computational Geometry 43(9), 713–720 (2010)
7. Kawamura, A., Matoušek, J., Tokuyama, T.: Zone diagrams in Euclidean spaces and in other normed spaces. Mathematische Annalen 354(4), 1201–1221 (2012)
8. Monterde, J., Ongay, F.: The distance trisector curve is transcendental. Geometriae Dedicata (in press)
9. Reem, D., Reich, S.: Zone and double zone diagrams in abstract spaces. Colloquium Mathematicum 115(1), 129–145 (2009)
10. Reem, D.: On the computation of zone and double zone diagrams. arXiv:1208.3124 (2012)

The Number of Different Unfoldings of Polyhedra[⋆]

Takashi Horiyama[1] and Wataru Shoji[2]

[1] Information Technology Center,
Saitama University, Saitama 338-8570, Japan
[2] Graduate School of Science and Engineering,
Saitama University, Saitama 338-8570, Japan
{horiyama,shoji}@al.ics.saitama-u.ac.jp

Abstract. Given a polyhedron, the number of its unfolding is obtained by the Matrix-Tree Theorem. For example, a cube has 384 ways of unfolding (i.e., cutting edges). By omitting mutually isomorphic unfoldings, we have 11 essentially different (i.e., nonisomorphic) unfoldings. In this paper, we address how to count the number of nonisomorphic unfoldings for any (i.e., including nonconvex) polyhedron. By applying this technique, we also give the numbers of nonisomorphic unfoldings of all regular-faced convex polyhedra (i.e., Platonic solids, Archimedean solids, Johnson-Zalgaller solids, Archimedean prisms, and antiprisms), Catalan solids, bipyramids and trapezohedra. For example, while a truncated icosahedron (a Buckminsterfullerene, or a soccer ball fullerene) has 375,291,866,372,898,816, 000 (approximately 3.75×10^{20}) ways of unfolding, it has 3,127,432,220, 939,473,920 (approximately 3.13×10^{18}) nonisomorphic unfoldings. A truncated icosidodecahedron has 21,789,262,703,685,125,511,464,767,107, 171,876,864,000 (approximately 2.18×10^{40}) ways of unfolding, and has 181,577,189,197,376, 045,928,994,520,239,942,164,480 (approximately 1.82×10^{38}) nonisomorphic unfoldings.

1 Introduction

An unfolding (also called an edge unfolding, a net or a development) of a polyhedron is a simple polygon obtained by cutting along the edges of the polyhedron and unfolding it into a plane. The cut edges of an edge unfolding of a polyhedron form a spanning tree of the 1-skeleton (i.e., the graph formed by the vertices and the edges) of the polyhedron (See, e.g., [11, Lemma 22.1.1]), and vice versa. Since Kirchhoff's matrix-tree theorem gives the number of spanning trees for any graph, we can obtain the number of unfoldings for any polyhedron. For example, a cube has 384 unfoldings (i.e., 384 ways of cut edges).

Different cut edges, however, may have isomorphic unfoldings. In Fig. 1, (a) and (b) have different cut edges (depicted in bold lines), while their unfoldings have the same shape depicted in (c). In actual, 24 unfoldings of a cube are isomorphic to Fig. 1(c). Later in this paper, we consider two cases for counting the

[⋆] A preliminary version was presented at EuroCG2013.

number of unfoldings. (1) The number of *labeled unfoldings:* edges have labels and we distinguish unfoldings according to their cut edges. (2) The number of *nonisomorphic unfoldings:* we identify isomorphic unfoldings even if they have different edge labels. (We identify mirror images as isomorphic.) The 384 labeled unfoldings of a cube are classified into 11 essentially different (i.e., nonisomorphic) unfoldings.

As mentioned later in related work, the number of unfoldings are of great interest for their wide area of applications. As for the counting for concrete polyhedra, most of the results are on the numbers of labeled unfoldings, since they are obtained by the matrix-tree theorem. On the other hand, few are on the numbers of nonisomorphic unfoldings (e.g., those of Platonic solids [4,13,15], and Archimedean n-gonal prism with $n = 3$ to 14 [19]), while they also have rich store of mathematical knowledge.

Our Contribution. In this paper, we address how to count the number of nonisomorphic unfoldings for any polyhedron. The naive way for counting nonisomorphic unfoldings is to enumerate all labeled unfoldings and to omit isomorphic unfoldings. Unfortunately, a dodecahedron and an icosahedron have 5,184,000 labeled unfoldings, respectively, and the test for the isomorphism is tough. (The test may be required $\binom{5,184,000}{2}$ times.) We here note that unfoldings in this paper may have overlaps. (In [14], some overlapping unfoldings of some Archimedean solids are given.) Similar story happened on Platonic solids: When the numbers of their nonisomorphic unfoldings were first obtained in about 40 years ago [4,13,15], we could not distinguish whether they are overlapping or not. In quite recent years, by enumerating all unfoldings, it is proved that any edge unfolding of Platonic solids is a flat nonoverlapping simple polygon [14,18].

For counting the number of nonisomorphic unfoldings, we follow the basic idea in [13,15], that is, to use Burnside's lemma [8]: given a polyhedron P, the number of nonisomorphic unfoldings is obtained by $u(\Gamma) = \frac{1}{|\mathrm{Aut}\,\Gamma|}\sum_{g\in \mathrm{Aut}\,\Gamma} |\{T \in \mathcal{T} \mid T = gT\}|$, where Γ is the 1-skeleton of P, $u(\Gamma)$ is the number of nonisomorphic spanning trees of Γ, $\mathrm{Aut}\,\Gamma$ is the automorphism group of Γ, g is a permutation in $\mathrm{Aut}\,\Gamma$, and $\mathcal{T}$ is the set of spanning trees of Γ. Although we can compute this equation by checking $T = gT$ (i.e., T has the same structure with the permuted tree gT) for every $g \in \mathrm{Aut}\,\Gamma$ and $T \in \mathcal{T}$, it is impractical for large $\mathcal{T}$ (e.g., $|\mathcal{T}| \simeq 2.18 \times 10^{40}$ for a truncated icosidodecahedron).

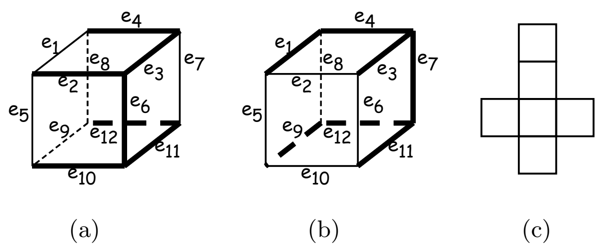


Fig. 1. Different cut edges (a) and (b) have isomorphic unfoldings

To overcome this situation, we follow the second idea given in [7], that is, to use a quotient graph. A quotient graph $\mathcal{Q}(\Gamma, g)$ is intuitively obtained by iterative contraction of two vertices u and v in Γ satisfying $u = g(v)$ (the edges are also contracted in the similar manner). By analyzing the structure of $\mathcal{Q}(\Gamma, g)$, we can obtain $|\{T \in \mathcal{T} \mid T = gT\}|$ without checking $T = gT$ for each $T \in \mathcal{T}$. In [7], they analyzed three cases which is necessary to obtain the number of nonisomorphic unfoldings of the five Platonic solids (in 3-dimensions) and the six regular convex polytopes in 4-dimensions. In this paper, we extend the technique so that we can apply it to any polyhedron.

Another contribution of this paper is the numbers of nonisomorphic unfoldings of all regular-faced convex polyhedra (i.e., Platonic solids, Archimedean solids, Johnson-Zalgaller solids, Archimedean prisms, and antiprisms). Furthermore, the numbers of nonisomorphic unfoldings of Catalan solids, bipyramids and trapezohedra are obtained, since they are the duals of Archimedean solids, prisms and antiprisms. For example, while a truncated icosahedron (and also a pentakis dodecahedron) has 375,291,866,372,898,816,000 (approximately 3.75×10^{20}) labeled unfoldings, it has 3,127,432,220,939,473,920 (approximately 3.13×10^{18}) nonisomorphic unfoldings. A truncated icosidodecahedron (and also a disdyakis triacontahedron) has 21,789,262,703,685,125,511,464,767,107,171,876, 864,000 (approximately 2.18×10^{40}) labeled unfolding, and has 181,577,189,197, 376,045,928,994,520,239,942,164,480 (approximately 1.82×10^{38}) nonisomorphic unfoldings.

Related Work. Table 1 gives the number of unfoldings of Platonic solids [4,13,15]. In computational chemistry, fullerenes are of interest. Buckminsterfullerene (also known as icosahedral C_{60}, soccerballene, or truncated icosahedron) has 375,291,866,372,898,816,000 labeled unfoldings [5]. The numbers of labeled unfoldings of Handballene (also know as truncated dodecahedral C_{60}, or truncated dodecahedron) and Archimedean (also known as truncated icosidodecahedral C_{120}, or truncated icosidodecahedron) are given in [6]. In [18], the number of nonisomorphic unfoldings of a truncated octahedron is estimated to be approximately 2,300,000. Akiyama et al. [1] are interested in the tessellation by unfoldings of polyhedra with regular polygonal faces, and the number of labeled unfoldings are investigated in their first step. [1] gives the number of labeled unfoldings of a cubotahedron, a truncated tetrahedron, and 17 out of 92 Johnson-Zalgaller solids. Archimedean prisms and Archimedean antiprisms are also of in-

Table 1. The number of unfoldings of Platonic solids

Name	#(Labeled unfoldings)	#(Nonisomorphic unfoldings)
Tetrahedron	16	2
Cube	384	11
Octahedron	384	11
Dodecahedron	5,184,000	43,380
Icosahedron	5,184,000	43,380

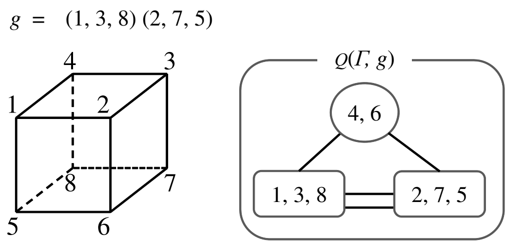


Fig. 2. Quotient graph $\mathcal{Q}(\Gamma, g)$ for a rotation $g = (1, 3, 8)(2, 7, 5)$

terest. n-gonal prism and n-gonal antiprism has $\frac{n}{2}\{(2+\sqrt{3})^n+(2-\sqrt{3})^n-2\}$ [3] and $\frac{2n}{5}\{(2+\sqrt{3})^n+(2-\sqrt{3})^n-2\}$ [16] labeled unfoldings, respectively. As for the number of their nonisomorphic unfoldings, only the cases for n-gonal prism with $n = 3$ to 14 are known [19].

2 Preliminaries

Let Γ, $V(\Gamma)$, $E(\Gamma)$, $\mathrm{Aut}\,\Gamma$, $\mathcal{T}(\Gamma)$ (or $\mathcal{T}$ in short) and $\mathcal{T}_g$ for $g \in \mathrm{Aut}\,\Gamma$, respectively, denote the 1-skeleton of a given polyhedron P, the set of vertices of Γ, the set of edges of Γ, the (symmetry) automorphism group of Γ, the set of spanning trees of Γ, and $\{T \in \mathcal{T} \mid T = gT\}$. Given a permutation $g \in \mathrm{Aut}\,\Gamma$, Fix g is the subgraph of Γ on which g acts as the identity. In other words, the edges and vertices of Fix g is defined as $V(\mathrm{Fix}\ g) = \{v \in V(\Gamma) \mid g(v) = v\}$ and $E(\mathrm{Fix}\ g) = \{(u, v) \in E(\Gamma) \mid u, v \in V(\mathrm{Fix}\ g)\}$. If g has no fixed point, i.e., $V(\mathrm{Fix}\ g) = \emptyset$, we use notation Fix $g = \emptyset$. Let $\alpha(g)$ denote the number of g-invariant edges in Γ, i.e., $|\{(u, v) \in E(\Gamma) \mid (u, v) = (g(u), g(v))$ or $(u, v) = (g(v), g(u))\}|$.

The quotient graph $\mathcal{Q}(\Gamma, g)$ is defined as follows. The orbit θ_v of vertex $v \in V(\Gamma)$ is the set of vertices $\{u \in V(\Gamma) \mid u = g^n(v), n \in \mathbb{Z}\}$. Let Ω be the set of orbits of length > 1. Then, the set U of vertices of $\mathcal{Q}(\Gamma, g)$ is defined as $U = \Omega$ if Fix $g = \emptyset$ holds, otherwise $U = \Omega \cup \{V(\mathrm{Fix}\ g)\}$. Let $\pi : V \to U$ be the natural projection defined as $\pi(v) = V(\mathrm{Fix}\ g)$ if $v \in V(\mathrm{Fix}\ g)$, otherwise $\pi(v) = \theta_v$. Let E' denote the set $\{(u, v) \in E(\Gamma) \mid \pi(u) \neq \pi(v)\}$. The projection π induces a map: $\tilde{\pi} : E' \to \mathcal{F} : (u, v) \to (\pi(u), \pi(v))$. The quotient graph $\mathcal{Q}(\Gamma, g)$ is defined as multigraph $(U, \mathcal{F})$, in which all edges in the same orbit of E' corresponds to an edge in $\mathcal{F}$.

Fig. 2 illustrates the quotient graph $\mathcal{Q}(\Gamma, g)$ for a rotation $g = (1, 3, 8)(2, 7, 5)$. The order of g is 3. We can observe that $V(\mathrm{Fix}\ g) = \{4, 6\}$, $E(\mathrm{Fix}\ g) = \emptyset$. Since we have two orbits $\{1, 3, 8\}$ and $\{2, 7, 5\}$, the set of vertices of $\mathcal{Q}(\Gamma, g)$ is $U = \{\{1, 3, 8\}, \{2, 7, 5\}, \{4, 6\}\}$. Since we have two orbits of edges $\{(1, 2), (3, 7), (8, 5)\}$ and $\{(1, 5), (3, 2), (8, 7)\}$, $\mathcal{Q}(\Gamma, g)$ has two edges between $\{1, 3, 8\}$ and $\{2, 7, 5\}$.

In [7], in order to obtain $|\mathcal{T}_g|$ for any g of regular convex polytopes in dimension ≤ 4, the following three facts are used:

Theorem 1 ([7])
(1) Let g be of prime order p and let $\mathrm{Fix}\ g \neq \emptyset$. Then, $|\mathcal{T}_g| = |\mathcal{T}(\mathrm{Fix}\ g)| \cdot |\mathcal{T}(\mathcal{Q}(\Gamma, g))|$.
(2) Let g be of order 2 with $\mathrm{Fix}\, g = \emptyset$. Then, $|\mathcal{T}_g| = |\mathcal{T}(\mathcal{Q}(\Gamma, g))| \cdot \alpha(g)$.
(3) If $\mathrm{Fix}\, g \neq \emptyset$ and $\mathrm{Fix}\, g$ is not connected, $|\mathcal{T}_g| = 0$ holds.

3 The Number of Nonisomorphic Unfoldings

We restrict Theorem 1 in 3-dimensions, and then extend it so as to obtain $|\mathcal{T}_g|$ for any g of any polyhedron. We first extend Theorem 1 by extending the notion of the quotient graph. Then, we discuss the (symmetry) automorphism groups of polyhedra. Our extended theorem consists of the following four facts. The first one is a straightforward extension of Theorem 1(1) in 3-d (note that if g is of prime order p then all vertices not in $V(\mathrm{Fix}\ g)$ have orbits of length p), and the fourth one is new. We also note here that Theorem 1 itself does not assume the convexity of a given polyhedron, neither does Theorem 2.

Theorem 2
(1) Let $\mathrm{Fix}\, g \neq \emptyset$ and all vertices not in $V(\mathrm{Fix}\, g)$ have orbits of the same length. Then, $|\mathcal{T}_g| = |\mathcal{T}(\mathrm{Fix}\, g)| \cdot |\mathcal{T}(\mathcal{Q}(\Gamma, g))|$.
(2) Let g be of order 2 with $\mathrm{Fix}\, g = \emptyset$. Then, $|\mathcal{T}_g| = |\mathcal{T}(\mathcal{Q}(\Gamma, g))| \cdot \alpha(g)$.
(3) If $\mathrm{Fix}\, g \neq \emptyset$ and $\mathrm{Fix}\, g$ is not connected, $|\mathcal{T}_g| = 0$ holds.
(4) If $\mathrm{Fix}\, g = \emptyset$ and $\alpha(g) = 0$, $|\mathcal{T}_g| = 0$ holds.

The proof of Theorem 2 starts with the following lemmas concerning the cardinality of the orbits.

Lemma 1. *Let T be a spanning tree in $\mathcal{T}_g$, and three vertices u, v, v' ($\in V(T)$) satisfy $u, v, v' \notin V(\mathrm{Fix}\ g)$, $\theta_u \neq \theta_v$, $|\theta_u| = |\theta_v|$ and $\theta_v = \theta_{v'}$. Then, two edges (u, v) and (u, v') cannot be in $E(T)$ at the same time.*

Proof. Suppose that (u, v) and (u, v') are included in $T \in \mathcal{T}_g$. Let k, ℓ be integers satisfying $g^k(v) = v'$ and $g^{k\ell}(v) = v$, respectively, and g' denote g^k. Then, we have a path $v, u, g'(v)$ in T. By applying g'^i to the path ($i = 1, 2, \dots \ell - 1$), we have another path $g'^i(v), g'^i(u), g'^{i+1}(v)$ in T. The concatenation of those paths is a cycle, which contradicts the definition of spanning tree T. □

Lemma 2. *Let T be a spanning tree in $\mathcal{T}_g$, and two vertices u, v ($\in V(T)$) satisfy $|\theta_u| \bmod |\theta_v| \neq 0$ and $|\theta_v| \bmod |\theta_u| \neq 0$. Then, $(u, v) \notin E(T)$ holds.*

Lemma 3. *Let T be a spanning tree in $\mathcal{T}_g$, and three vertices u, v, v' ($\in V(T)$) satisfy $u, v, v' \notin V(\mathrm{Fix}\, g)$, $\theta_u \neq \theta_v$ and $\theta_v = \theta_{v'}$. If two edges (u, v_0) and (u, v_1) are in $E(T)$, then $|\theta_v| > |\theta_u|$ holds.*

The following lemma gives the converse of Lemma 3

Lemma 4. *Let T be a spanning tree in $\mathcal{T}_g$, and two vertices u, v $(\in V(T))$ satisfy $(u,v) \in E(T)$ and $|\theta_u| < |\theta_v|$. Then, for any u' in θ_u, there exist two vertices $v', v'' \in \theta_v$ satisfying that two edges (u',v') and (u',v'') are in T.*

Next, we consider a path in Γ, whose length is at least 2. Let $p(v_0, v_k)$ denote a path $v_0, v_1, \ldots, v_k$. Then, we have the following lemmas.

Lemma 5. *Let $p(v_0, v_k)$ be a path satisfying $|\theta_{v_0}| < |\theta_{v_1}| \leq |\theta_{v_2}| \leq \cdots \leq |\theta_{v_{k-1}}| > |\theta_{v_k}|$, where $k \geq 2$. Then, $p(v_0, v_k)$ can not be included in any $T \in \mathcal{T}_g$.*

Proof. Suppose that $p(v_0, v_k)$ is in some T. From Lemma 4, $|\theta_{v_0}| < |\theta_{v_1}|$ implies that any vertex in θ_{v_0} has at least two edges in T connecting with some vertices in θ_{v_1}. Similarly, any vertex in θ_{v_k} has at least two edges in T connecting with some vertices in $\theta_{v_{k-1}}$. Any vertex in θ_{v_i} $(1 \leq i < k)$ has an edge in T connecting with a vertex in $\theta_{v_{i-1}}$ and an edge connecting with a vertex in $\theta_{v_{i+1}}$. This observation implies that T has a cycle, which is a contradiction. □

Lemma 6. *Let g satisfy $\alpha(g) = 0$, and two vertices u and v have the same orbit, i.e., $\theta_u = \theta_v$. Then, edge (u,v) can not be included in any $T \in \mathcal{T}_g$.*

Proof. Suppose that (u,v) is in some T. $\alpha(g) = 0$ implies $|\theta_u| > 2$. Let k, ℓ be integers satisfying $g^k(v) = u$ and $g^{k\ell}(v) = v$, respectively, and g' denote g^k. By applying g'^i $(i = 1, 2, \ldots, \ell - 1)$ to edge $(v, g'(v))$, we have edge $(g'^i(v), g'^{i+1}(v))$ in $T \in \mathcal{T}_g$, which implies T has a cycle of length ℓ. □

Lemma 7. *Let $p(v_0, v_k)$ be a path of length at least 2 $(k \geq 2)$, and the quotient graph $\mathcal{Q}(\Gamma, g)$ satisfy $\pi(v_0) = \pi(v_k)$ and $\pi(v_1) \neq \pi(v_{k-1})$. Then, $p(v_0, v_k)$ can not be included in any $T \in \mathcal{T}_g$.*

Proof. Suppose that $p(v_0, v_k)$ is in some T. $\pi(v_0) = \pi(v_k)$ implies the following two cases. Case (1): Both v_0 and v_k are fixed. $\pi(v_1) \neq \pi(v_{k-1})$ implies that at least one of v_1 and v_{k-1} is not fixed. W.l.o.g., we suppose v_1 is not fixed. Then, $|\theta_{v_1}| > 1 = |\theta_{v_0}| = |\theta_{v_k}|$ holds. Thus, there exists the minimum k' $(1 \leq k' < k)$ satisfying $|\theta_{v_{k'}}| > |\theta_{v_{k'+1}}|$, which implies $|\theta_{v_0}| < |\theta_{v_1}| \leq \cdots \leq |\theta_{v_{k'}}| > |\theta_{v_{k'+1}}|$. Subpath $p(v_0, v_{k'+1})$ in T contradicts Lemma 5. Case (2): Neither v_0 nor v_k is fixed. We have $\theta_{v_0} = \theta_{v_k}$ and thus have a cycle in T. □

Proof of Theorem 2. Suppose that there exists a g-invariant spanning tree $T \in \mathcal{T}_g$. Let two vertices u_0 and v_0 have the same orbit, i.e., $\theta_{u_0} = \theta_{v_0}$. (Otherwise, all vertices are in $V(\text{Fix } g)$.) In T, path $p(u_0, v_0)$ is uniquely determined. From Lemma 6, the length of the path is at least 2.

Now, we relabel the vertices in $p(u_0, v_0)$. Let $\mathbf{u} = (u_0, u_1, \ldots, u_k)$ (respectively, $\mathbf{v} = (v_0, v_1, \ldots, v_k)$) denote the sequence of the vertices visited from u_0 to $v_0 = u_k$ (respectively, from v_0 to $u_0 = v_k$), where $k \geq 2$. Let $\mathbf{s_u} = (\theta_{u_0}, \theta_{u_1}, \ldots, \theta_{u_k})$ and $\mathbf{s_v} = (\theta_{v_0}, \theta_{v_1}, \ldots, \theta_{v_k})$ denote the sequences of the orbits of $\mathbf{u}$ and $\mathbf{v}$, respectively. Then, we have the following two cases.

Case (1): $\mathbf{s_u} \neq \mathbf{s_v}$. Let i be the minimum integer satisfying $\theta_{u_i} = \theta_{v_i}$ and $\theta_{u_{i+1}} \neq \theta_{v_{i+1}}$. From Lemma 7, subpath $p(u_i, v_i)$ cannot be in T, which contradicts the definition of $p(u_0, v_0)$.

Case (2): $\mathbf{s_u} = \mathbf{s_v}$. If k is odd, $\mathbf{s_u} = \mathbf{s_v}$ implies $\theta_{u_{\lfloor k/2 \rfloor}} = \theta_{v_{\lfloor k/2 \rfloor}}$. Since $v_{\lfloor k/2 \rfloor} = u_{\lceil k/2 \rceil}$, edge $(u_{\lfloor k/2 \rfloor}, v_{\lfloor k/2 \rfloor})$ is in T, which contradicts Lemma 6. Thus, k is even, and $u_{k/2} = v_{k/2}$ holds. Since T has edges $(u_{k/2}, u_{k/2-1})$, $\theta_{u_{k/2-1}} \neq \theta_{u_{k/2}}$ holds from Lemma 6. Since T has two edges $(u_{k/2}, u_{k/2-1})$, and $\theta_{u_{k/2-1}} = \theta_{u_{k/2+1}}$ holds, we have $|\theta_{u_{k/2-1}}| > |\theta_{u_{k/2}}|$ from Lemma 3. If $|\theta_{u_i}| < |\theta_{u_{i+1}}| \leq \ldots \leq |\theta_{u_{k/2-1}}| > |\theta_{u_{k/2}}|$ holds for some i $(0 \leq i < k/2 - 1)$, subpath $p(u_i, u_{k/2})$ in T contradicts Lemma 5. Thus, we have $|\theta_{u_0}| \geq |\theta_{u_1}| \geq \cdots \geq |\theta_{u_{k/2-1}}| > |\theta_{u_{k/2}}|$.

Here, $u_{k/2}$ is not fixed since Fix $g = \emptyset$, which implies that $\theta_{u_{k/2}}$ has another vertex w. We regard $u_{k/2}$ and w as new u_0 and v_0, and recursively apply the above argument to the new vertices. $|\theta_{u_0}|$ monotonically decreases during the recursion, and the recursion continues while $|\theta_{u_0}| \geq 2$. Since the number of vertices are finite, the recursion terminates with $|\theta_{u_0}| = 1$, which contradicts Fix $g = \emptyset$. Thus, we have the lemma. □

In the rest of this section, we show that $|\mathcal{T}_g|$ can be obtained by any permutation in the symmetry group $\operatorname{Aut} \Gamma$ of any (i.e., including nonconvex) polyhedra.

Theorem 3. *Let P be a polyhedron, and $\operatorname{Aut} \Gamma$ be the symmetry group of P. Then, for any permutation $g \in \operatorname{Aut} \Gamma$, either of the four cases in Theorem 2 holds.*

Before the proof of the above theorem, we introduce two important lemmas.

Lemma 8 (see e.g., [9], p.312). *A polyhedron can have one of the following 17 types of symmetry*[1]*: C_1, C_i, C_s, C_n, C_{nv}, C_{nh}, D_n, D_{nh}, D_{nv} (also known as D_{nd}), S_n, T, T_d, T_h, O, O_h, I, I_h.*

Lemma 9 ([2]). *A symmetry group of a polyhedron is one of the above 17 types, and consists of the following permutations: (1) Identity: E. (2) Rotation: C_n^j. (3) Reflection: $\sigma_h, \sigma_v, \sigma_d, \sigma$. (4) Rotation-reflection: S_n^j. (5) Inversion: i.*

For example, if a polyhedron has the identity symmetry, but no other symmetries, its corresponding $\operatorname{Aut} \Gamma$ is C_1, i.e., $\operatorname{Aut} \Gamma = \{E\}$. As for a truncated dodecahedron, its corresponding $\operatorname{Aut} \Gamma$ is I_h, which consists of the identity, 59 rotations, 59 rotation-reflections, and the inversion.

Proof of Theorem 3. Now, we confirm that we can apply Theorem 2 to those five types of permutations. **Case (1):** if g is the identity, all vertices are fixed, and thus $|\mathcal{T}_g|$ is equivalent to $\mathcal{T}(\Gamma)$, i.e., the number of spanning trees of Γ. **Case (2):** if g is a rotation C_n^j, all vertices v not in Fix g are rotated by $360j/n$ degrees around the rotation-axis. The orbits of such vertices are of the same length. If Fix $g \neq \emptyset$, we can apply Theorem 2(1). Otherwise, since there are no g-invariant edges, we can apply Theorem 2(2). **Case (3):** if g is a reflection, by a similar argument with Case (2), we can apply Theorem 2(1) or (2).

Case (4): Rotation-reflection is a combination of a rotation with a reflection through a plane perpendicular to the rotation-axis. By the rotation, all vertices

[1] The prismatic types are discussed in the next section.

and edges not on the rotation-axis cannot remain in their original positions. Since the reflection-plane perpendicular to the rotation-axis, by the reflection, those cannot back to their original positions. Similarly, all vertices and edges on the rotation-axis cannot keep their original positions. Thus, we have Fix $g = \emptyset$ and $\alpha(g) = 0$, which implies we can apply Theorem 2.

Case (5): If g is the inversion, all vertices cannot remain in their original positions, which implies Fix $g = \emptyset$. By the inversion, no edges are g-invariant. (Otherwise, the edges go through the center of the polyhedron.) Thus, we can apply Theorem 2. □

4 Regular-Faced Convex Polyhedra

In this section, we apply the proposed method to all regular-faced convex polyhedra, i.e., the 5 Platonic solids, the 13 Archimedean solids, the 92 Johnson-Zalgaller solids, n-gonal Archimedean prisms, and n-gonal Archimedean antiprisms. We can also obtain the numbers of nonisomorphic unfoldings of Catalan solids, bipyramids and trapezohedra, since they are the duals of Archimedean solids, prisms, antiprisms. The adjacency matrices of regular-faced convex polyhedra are obtained from [17].

As for the Platonic solids, we have the same result with Table 1. The results for the Archimedean solids and the Johnson-Zalgaller solids are listed in Tables 2 and 3. In the tables, the entries without citation are newly obtained in this paper. A truncated icosahedron (and also a pentakis dodecahedron) has approximately 3.75×10^{40} labeled unfoldings, and has approximately 3.13×10^{38} nonisomorphic unfoldings (see Table 2 for their exact numbers). A truncated icosidodecahedron (and also a disdyakis triacontahedron) has approximately 2.18×10^{40} labeled unfolding, and has approximately 1.82×10^{38} nonisomorphic unfoldings. We also note here that the number of nonisomorphic unfoldings of a truncated octahedron is estimated to be approximately 2,300,000 [18], and the actual number is 2,108,512. Thus, the estimation is a good approximation.

Now, we consider an n-gonal Archimedean prism. For a cube (i.e., $n = 4$), we adopt $u(\Gamma) = 11$. For other prisms, we have the following theorem. (We have $4n$ permutations including the identity. For the identity, we can use $\mathcal{T}(\Gamma)$ in [3].)

Theorem 4. *The number of nonisomorphic unfoldings of an n-gonal Archimedean prism is*

$$u(\Gamma) = \begin{cases} \frac{1}{8\sqrt{3}} \left\{2\sqrt{3}\,n + \sqrt{3}(2+\sqrt{3})^n + (2+\sqrt{3})^{\lfloor \frac{n}{2} \rfloor}(4+2\sqrt{3}) \right. \\ \qquad \left. + (2-\sqrt{3})^{\lfloor \frac{n}{2} \rfloor}(2\sqrt{3}-4) + \sqrt{3}((2-\sqrt{3})^n - 2)\right\} & (n \text{ is odd}), \\ \frac{1}{24}\left\{6n + 3(2+\sqrt{3})^n + 4\sqrt{3}(2+\sqrt{3})^{\frac{n}{2}} \right. \\ \qquad \left. - 4\sqrt{3}(2-\sqrt{3})^{\frac{n}{2}} + 3(2-\sqrt{3})^n - 6\right\} & (n \text{ is even}). \end{cases}$$

By a similar argument as above, we can also obtain the number of nonisomorphic unfoldings of an n-gonal Archimedean antiprism. For an octahedron (i.e., $n = 3$), we adopt $u(\Gamma) = 11$. For other antiprisms, we have the following theorem. (We have $4n$ permutations. For the identity, we can use $\mathcal{T}(\Gamma)$ in [16].)

Table 2. The number of edge unfoldings of Archimedean solids, where the entries without citation are newly obtained in this paper

Name	#(Labeled unfoldings)	#(Nonisomorphic unfoldings)
Cuboctahedron	331,776 [1]	6,912
Icosidodecahedron	208,971,104,256,000	1,741,425,868,800
Truncated tetrahedron	6,000 [1]	261
Truncated octahedron	101,154,816	2,108,512
Truncated cube	32,400,000	675,585
Truncated icosahedron	375,291,866,372,898,816,000 [5]	3,127,432,220,939,473,920
Truncated dodecahedron	4,982,259,375,000,000,000 [6]	41,518,828,261,687,500
Rhombicuboctahedron	301,056,000,000	6,272,012,000
Rhombicosidodecahedron	201,550,864,919,150,779,950,956,544,000	1,679,590,540,992,923,166,257,971,200
Truncated cuboctahedron	12,418,325,780,889,600	258,715,122,137,472
Truncated icosidodecahedron	21,789,262,703,685,125,511,464,767,107,171,876,864,000 [6]	181,577,189,197,376,045,928,994,520,239,942,164,480
Snub cube	89,904,012,853,248	3,746,001,752,064
Snub dodecahedoron	438,201,295,386,966,498,858,139,607,040,000,000	7,303,354,923,116,108,380,042,995,304,896,000

Table 3. The number of edge unfoldings of Johnson-Zalgaller solids, where the entries without citation are newly obtained in this paper.

Name	#(Labeled unfoldings)	#(Nonisomorphic unfoldings)	Name	#(Labeled unfoldings)	#(Nonisomorphic unfoldings)
J01	45	8	J47	9,324,488,558,669,593,960	1,864,897,711,733,918,792
J02	121	15	J48	2,670,159,599,304,760,178,000	267,015,959,942,030,583,130
J03	1,815 [1]	308	J49	672	173
J04	24,000	3,030	J50	5,544	1,401
J05	297,025	29,757	J51	42,336	3,549
J06	78,250,050	7,825,005	J52	16,744	4,201
J07	361 [1]	63	J53	153,816	38,526
J08	3,509	448	J54	75,973 [1]	19,035
J09	30,976	3,116	J55	709,632 [1]	88,776
J10	27,216	3,421	J56	707,232 [1]	176,967
J11	403,202	40,321	J57	6,531,840 [1]	544,680
J12	75	9	J58	92,724,962	9,272,497
J13	1,805	99	J59	1,651,482,010	82,580,526
J14	1,728	156	J60	1,641,317,568	410,335,964
J15	31,500	2,010	J61	28,745,798,400	4,790,966,400
J16	508,805	25,574	J62	28,080	7,050
J17	207,368	13,041	J63	1,734	289
J18	1,609,152 [1]	268,260	J64	8,450	1,409
J19	227,402,340	28,427,091	J65	1,245,456 [1]	207,576
J20	29,821,320,745	2,982,139,245	J66	54,921,311,280	6,865,163,910
J21	8,223,103,375,490	822,310,337,549	J67	90,974,647,120,896	5,685,916,514,256
J22	37,158,912 [1]	6,193,152	J68	68,495,843,558,495,480,625,000	6,849,584,355,849,548,062,500
J23	15,482,880,000	1,935,360,000	J69	936,988,158,859,771,579,003,317,600	46,849,407,942,992,327,926,343,838
J24	5,996,600,870,820	599,660,087,082	J70	930,303,529,996,712,062,599,302,400	232,575,882,499,181,854,544,317,560
J25	1,702,422,879,696,000	170,242,287,969,600	J71	12,479,653,904,364,665,921,377,091,740,032	2,079,942,317,394,110,986,896,181,956,672
J26	1,176 [1]	152	J72	206,686,735,580,507,426,149,463,308,960	20,668,673,558,050,742,614,946,330,896
J27	324,900 [1]	27,195	J73	211,950,222,127,067,401,293,093,928,960	10,597,511,106,353,370,064,654,696,448
J28	29,859,840 [1]	1,867,560	J74	211,595,653,377,414,999,219,839,524,608	52,898,913,344,353,749,804,959,881,152
J29	30,950,832 [1]	1,934,427	J75	216,255,817,875,464,148,759,178,607,616	36,042,636,312,577,358,126,529,767,936
J30	2,518,646,460	125,939,163	J76	21,081,520,904,394,872,104,529,280	2,108,152,090,439,487,210,452,928
J31	2,652,552,060	132,627,603	J77	21,635,458,027,234,604,842,992,000	2,163,545,802,723,460,484,299,200
J32	699,537,024,120	69,953,702,412	J78	21,638,184,348,166,814,636,938,752	10,819,092,174,083,407,318,469,376
J33	745,208,449,920	74,520,844,992	J79	22,171,247,351,297,062,278,807,776	11,085,623,675,648,531,139,403,888
J34	193,003,269,869,040	9,650,165,403,136	J80	2,163,645,669,729,922,583,040	108,182,283,486,496,129,152
J35	301,896,210 [1]	25,158,925	J81	2,094,253,294,125,015,611,392	523,563,323,531,253,902,848
J36	302,400,000 [1]	25,203,000	J82	2,151,245,812,763,713,106,752	1,075,622,906,381,856,553,376
J37	301,988,758,680	18,874,379,520	J83	197,148,908,795,401,104	32,858,151,465,900,184
J38	270,745,016,304,350	13,537,250,963,730	J84	8,640	1,109
J39	272,026,496,000,000	13,601,327,004,000	J85	1,291,795,320 [1]	80,742,129
J40	75,378,202,163,880,700	7,537,820,216,388,070	J86	84,480	21,204
J41	75,804,411,381,317,500	7,580,441,138,131,750	J87	652,846	326,423
J42	20,969,865,292,417,385,400	1,048,493,264,659,994,295	J88	2,002,440	500,959
J43	21,115,350,368,078,435,000	1,055,767,519,017,973,725	J89	32,373,600	8,094,150
J44	5,295,528,588 [1]	882,609,105	J90	519,556,800	64,950,268
J45	13,769,880,349,680	1,721,235,971,518	J91	870,912	108,936
J46	32,543,644,773,848,180	3,254,364,517,723,165	J92	235,726,848	39,287,808

Theorem 5. *The number of nonisomorphic unfoldings of an n-gonal Archimedean antiprism is*

$$u(\Gamma) = \frac{1}{10}\left\{\left(\tfrac{1+\sqrt{5}}{2}\right)^{4n} + \left(\tfrac{1+\sqrt{5}}{2}\right)^{-4n} - 2\right\} + \frac{(3+\sqrt{5})^n-(3-\sqrt{5})^n}{2^{n+1}\sqrt{5}}.$$

References

1. Akiyama, J., Kuwata, T., Langerman, S., Okawa, K., Sato, I., Shephard, G.C.: Determination of All Tessellation Polyhedra with Regular Polygonal Faces. In: Akiyama, J., Bo, J., Kano, M., Tan, X. (eds.) CGGA 2010. LNCS, vol. 7033, pp. 1–11. Springer, Heidelberg (2011)
2. Atkins, P.W., Child, M.S., Phillips, C.S.G.: Tables for Group Theory, Oxford University Press (1970)
3. Boesch, G.F.T., Bogdanowicz, Z.R.: The Number of Spanning Trees in a Prism, Inter. J. Comput. Math. 21, 229–243 (1987)
4. Bouzette, S., Vandamme, F.: The regular Dodecahedron and Icosahedron unfold in 43380 ways (unpublished manuscript)
5. Brown, T.J.N., Mallion, R.B., Pollak, P., de Castro, B.R.M., Gomes, J.A.N.F.: The number of spanning trees in buckminsterfullerene. Journal of Computational Chemistry 12, 1118–1124 (1991)
6. Brown, T.J.N., Mallion, R.B., Pollak, P., Roth, A.: Some Methods for Counting the Spanning Trees in Labelled Molecular Graphs, examined in Relation to Certain Fullerenes. Discrete Applied Mathematics 67, 51–66 (1996)
7. Buekenhout, F., Parker, M.: The Number of Nets of the Regular Convex Polytopes in Dimension ≤ 4. Disc. Math. 186, 69–94 (1998)
8. Burnside, A.: Theory of Groups of Finite Order. Cambridge University Press (1911)
9. Cromwell, P.R.: Polyhedra. Cambridge University Press (1997)
10. Coxeter, H.S.M.: Regular and semi-regular polytopes. II. Math. Z. 188, 3–45 (1985)
11. Demaine, E.D., O'Rourke, J.: Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press (2007)
12. Haghigh, M.H.S., Bibaki, K.: Recursive Relations for the Number of Spanning Trees. Applied Mathematical Sciences 3(46), 2263–2269 (2009)
13. Hippenmeyer, C.: Die Anzahl der inkongruenten ebenen Netze eines regulären Ikosaeders. Elem. Math. 34, 61–63 (1979)
14. Horiyama, T., Shoji, W.: Edge unfoldings of Platonic solids never overlap. In: Proc. of the 23rd Canadian Conference on Computational Geometry, pp. 65–70 (2011)
15. Jeger, M.: Über die Anzahl der inkongruenten ebenen Netze des Würfels und des regulären Oktaeders. Elemente der Mathematik 30, 73–83 (1975)
16. Kleitman, D.J., Golden, B.: Counting trees in a certain class of graphs. Am. Math. Monthly 82, 40–44 (1975)
17. Kobayashi, M., Suzuki, T.: Data of coordinates of all regular-faced convex polyhedra (1992), `http://mitani.cs.tsukuba.ac.jp/polyhedron/`
18. Pandey, S., Ewing, M., Kunas, A., Nguyen, N., Gracias, D.H., Menon, G.: Algorithmic design of self-folding polyhedra. Proc. Natl. Acad. Sci. USA 108(50), 19885–19890 (2011)
19. Sloane, N.J.A.: Sequence A103535, The On-Line Encyclopedia of Integer Sequences

Computing the Smallest Color-Spanning Axis-Parallel Square

Payam Khanteimouri[1], Ali Mohades[1],
Mohammad Ali Abam[2], and Mohammad Reza Kazemi[1]

[1] Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
[2] Sharif University of Technology, Tehran, Iran

Abstract. For a given set of n colored points with k colors in the plane, we study the problem of computing the smallest color-spanning axis-parallel square. First, for a dynamic set of colored points on the real line, we propose a dynamic structure with $O(\log^2 n)$ update time per insertion and deletion for maintaining the smallest color-spanning interval. Next, we use this result to compute the smallest color-spanning square. Although we show there could be $\Omega(kn)$ minimal color-spanning squares, our algorithm runs in $O(n \log^2 n)$ time and $O(n)$ space.

Keywords: Computational Geometry, Algorithm, Color-Spanning Objects.

1 Introduction

Background. Suppose there are k different types of facilities like banks, police offices, etc. and we are given n facilities of these types. A basic problem arising here is to find a region in which there is at least one representative from each type of facilities. This suggests the problems of computing *the smallest area/perimeter color-spanning objects*. A region is said to be *color-spanning* if it contains at least one point from each color. Another motivation comes from *discrete imprecise data*. In this context, each imprecise point is defined with a set of discrete possible locations. So, for a given k imprecise points we have a set of n points with k colors, where n is the number of possible locations for all imprecise points. The basic problem on a set of imprecise points is to locate each imprecise point within its defining set in which a measure becomes optimized [9]. This is equivalent to choose exactly k points with different colors in which a property e.g. diameter, closest pair, bounding box, etc. gets minimized or maximized. Beside these two applications, this problem has other applications in statistical clustering, pattern recognition and generalized range searching [7,10,12].

Related Works. There are several works on colored points. Motivated from imprecise data, Fan et al. [5] showed some results on hardness of *the largest closest pair* and *the minimum planar spanning tree*. Fan et al. [6] also proposed an $O(n^2 \log n)$ time algorithm to compute the expected area of convex hulls of the color-spanning sets. In addition, Consuegra et al. [3] considered the same setting and solved several other problems.

In the view of location problems, for a given set of n colored points with k colors in the plane, one of the most studied problems is to find *the smallest color-spanning rectangle.* For the axis-parallel case, Abellanas et al. [1] showed there are $\Theta\big((n-k)^2\big)$ minimal rectangles in the worst case and proposed an algorithm of $O\big((n-k)^2 \log^2 k\big)$ running time and $O(n)$ space to solve the problem. The algorithm has been recently improved to $O(n^2 \log n)$ time by Das et al. [4]. For arbitrary oriented case, Das et al. [4] proposed an algorithm in $O(n^3 \log k)$ time and $O(n)$ space. Furthermore, they solved the problem of computing *the smallest color-spanning strip* in $O(n^2 \log n)$ time and $O(n)$ space using the dual of the given points. The results are near efficient with respect to testing all minimal objects. A *minimal color-spanning object* contains at least one point from each color and any sub-region of it does not contain all colors.

For the problem of computing *the smallest color-spanning circle*, Abellanas et al. [2] proposed an algorithm with $O\big(n^2\alpha(k)\log k\big)$ time using *farthest colored Voronoi diagram (FCVD).* For a given set of colored points with k colors in the plane, the FCVD is a subdivision of the plane in which for any region R there is a unique site p such that any color-spanning circle centered at a point in R contains p. Indeed, a vertex of the FCVD which is adjacent to three regions with different colors of their sites, is a candidate of being the center of the smallest color-spanning circle. Therefore, a simple algorithm is to compute the FCVD and test circles centered at the vertices of FCVD. The other approach mentioned by Abellanas et al. [1] is to obtain the smallest color-spanning circle and the smallest color-spanning axis-parallel square in $O(kn \log n)$ time and $O(n)$ space using the upper envelope of Voronoi surfaces [8].

Our Results. In this paper, first we consider the problem of maintaining the smallest color-spanning interval for a dynamic set of colored points with k colors on the real line. We propose a data structure which spends $O(\log^2 n)$ update time per insertion and deletion using the structure designed by Overmars and van Leeuwen [11]. Next, we concentrate on the problem of computing the smallest color-spanning axis-parallel square. The algorithm sweeps the points and keeps the smallest color-spanning interval for the projection of the points on the sweep line when insertion or deletion occurs. We use the mentioned dynamic structure for maintaining the smallest color-spanning interval to solve the problem. Our algorithm runs in $O(n \log^2 n)$ time and $O(n)$ space. The algorithm does not test every minimal candidates (in fact, we show that there may be $\Theta(kn)$ minimal color-spanning axis-parallel squares in the worst case). So, this result is an improvement to the result proposed by Abellanas et al. [1] using the upper envelope of Voronoi surfaces [8].

This paper is organized as follows. In Section 2 we show how to maintain the smallest color-spanning interval for a dynamic set of colored points on the real line. Next, in Section 3 we propose an algorithm to compute the smallest color-spanning square using the results of Section 2. Finally we conclude in Section 4.

2 Dynamic Maintenance of Minimal Color-Spanning Intervals

In this section, we concentrate on color-spanning intervals for a set of colored points with k colors on the real line. For ease of the presentation, we assume that points are in general position which means point coordinates are different. We first show some properties of color-spanning intervals for a static set of colored points on the real line. Next, we consider the problem of maintaining the smallest color-spanning interval for dynamic points in which the points are permitted to be inserted or deleted.

2.1 Minimal Color-Spanning Intervals for Static Points

A minimal color-spanning interval for a set of colored points on the real line is an interval containing all colors and any sub-interval of it does not contain all colors. As a simple observation, the endpoints of a minimal color-spanning interval have different colors and their colors are unique in the interval.

Suppose we are given a set $\mathcal{P}$ of n colored points with k colors on the real line. It is easy to show that the number of minimal color-spanning intervals is linear and they can be found with a simple algorithm in linear time and space apart from sorting. The algorithm sweeps the points from left to right with two sweep lines which stop at the endpoints of an interval. It uses an array for keeping the number of points from each color and a variable for the number of different colors between the two sweep lines. Since the sweep lines never go back, the algorithm takes $O(n)$ time and space. We omit the details due to the simplicity and conclude the following theorem.

Theorem 1. *For a given set of n points with k colors on the real line, the smallest color-spanning interval can be computed in $O(n)$ time and space apart from sorting.*

In the following, we show a new view of minimal color-spanning intervals. Gupta et al. [7] use a transformation to perform generalized range reporting/counting for a given set of colored points on the real line. Indeed, they map the original given points on the real line to points in the transformed plane and perform the standard known 3-sided range reporting/counting in the transformed plane. We exploit the same transformation to give a new view of minimal color-spanning intervals. Let $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ be a given set of colored points with k colors on the real line. For each color c, first we sort the points with color c (c-colored points, for short) in an increasing order. Then, for an arbitrary point p_i with color c, let $pred(p_i)$ be the previous point of p_i in the list of the ordered c-colored points. In addition, we set $pred(p_j) = -\infty$ if p_j is the leftmost point with color c. Moreover, we insert k additional points from each color at $+\infty$—see Figure 1. A point p_i is mapped to the point $p_i^* = \big(p_i, pred(p_i)\big)$ in the transformed plane. Let $\mathcal{P}^* = \{p_1^*, p_2^*, \cdots, p_{n+k}^*\}$ be the transformed points—Figure 3 shows the transformed points of Figure 1. Furthermore, an interval $I = [l, r]$ on the real line is mapped to the point $I^* = (r, l)$. This transformation has several interesting properties.

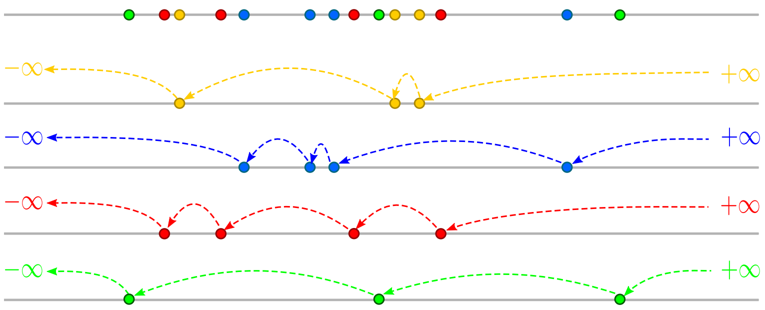


Fig. 1. The ordered lists of c-colored points for each color c

Consider an arbitrary point p in the plane. The vertical and the horizontal lines passing through p divide the plane into four quadrants. Let $\sigma(p)$ be the right-bottom quadrant; precisely $\sigma(p) = \{q \in \mathbb{R}^2 | p_x \leq q_x, p_y \geq q_y\}$. In the following we give some related definitions.

Definition 1. *For a set of points $\mathcal{Q} = \{q_1, \cdots, q_n\}$ and a point q in the plane, $\sigma(q)$ is empty with respect to $\mathcal{Q}$ if there is no point $q_i \in \mathcal{Q}$ in the interior of $\sigma(q)$.*

Definition 2. *For a set $\mathcal{Q}$ of points in the plane, a point $q \in \mathbb{R}^2$ is maximal with respect to $\mathcal{Q}$ if $\sigma(q)$ is empty and there is no other empty $\sigma(p)$ for some point $p \in \mathbb{R}^2$ such that $\sigma(q) \subset \sigma(p)$.*

Definition 3. *For a set of points $\mathcal{Q}$ in the plane, a point $q_i \in \mathcal{Q}$ is a skyline point if $\sigma(q_i)$ is empty.*

In order to see how the maximal points in the plane are related to the minimal color-spanning intervals, we present the following lemma.

Lemma 1. *For a given set $\mathcal{P}$ of colored points on the real line, $I = [l, r]$ is a minimal color-spanning interval if and only if the point $I^* = (r, l)$ is maximal with respect to the points $\mathcal{P}^*$ in the transformed plane.*

Proof. First, suppose $I = [l, r]$ is a minimal color-spanning interval. For the sake of contradiction, assume $I^* = (r, l)$ is not maximal. We distinguish two cases: (1) $\sigma(I^*)$ is not empty, (2) $\sigma(I^*)$ is empty but it is not maximal. In case (1) assume point $\big(p_i, pred(p_i)\big)$ is inside $\sigma(I^*)$ where both p_i and $pred(p_i)$ have color c—see Figure 2. This gives us $l > pred(p_i)$ and $r < p_i$ which means $[l, r]$ is a proper subinterval of $\big[pred(p_i), p_i\big]$. Therefore, $[l, r]$ does not contain any point of color c which is a contradiction. Now, consider case (2). In this case, there is a point $q^* = (r', l')$ such that $\sigma(I^*) \subset \sigma(q^*)$. This means there is a smaller color-spanning interval $q = [l', r']$ contained in $I = [l, r]$ which is a contradiction. Moreover, additional points at infinity makes any minimal color-spanning interval covered by maximal points in the transformed plane. The converse implication can be proved in a similar way. □

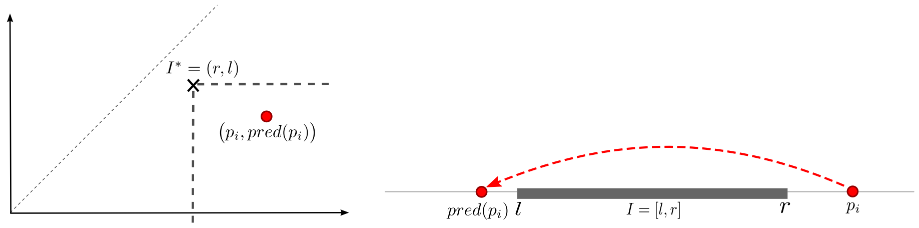
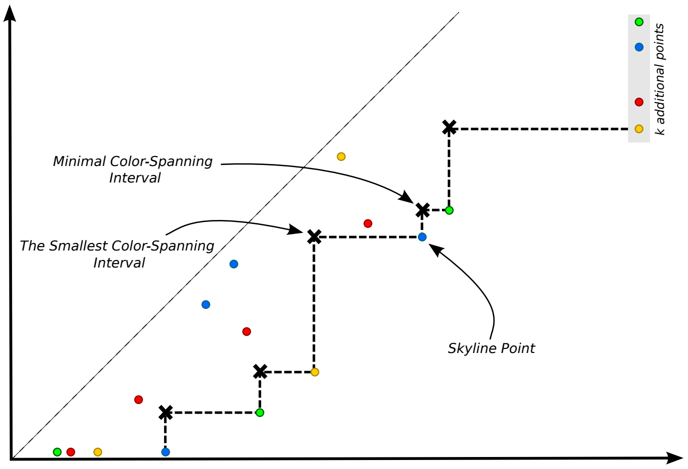


Fig. 2. A not color-spanning interval $I = [l, r]$ where $\sigma(I^*)$ is not empty

To summarize, a minimal color-spanning interval I is a maximal point in the transformed plane. In fact, vertical and horizontal rays extending to $+y$ and $-x$ directions starting at skyline points define the maximal points—see Figure 3 for more illustration. Consider I^* as a maximal point between two consecutive skyline points $p^*_{c_i} = (p_{c_i}, p_{c_{i-1}})$ and $p^*_{c'_j} = (p_{c'_j}, p_{c'_{j-1}})$ in the transformed plane. The minimal color-spanning interval defined by I^* is $I = [p_{c'_{j-1}}, p_{c_i}]$. Furthermore, the length of an interval is the vertical distance of its transformed point to the line $y = x$. The smallest color-spanning interval is *the minimum maximal point* in the transformed plane—see Figure 3.



k additional points
Minimal Color-Spanning Interval
The Smallest Color-Spanning Interval
Skyline Point

Fig. 3. The smallest color-spanning interval on the staircase of maximal points

2.2 Minimal Color-Spanning Intervals for Dynamic Points

In this section, we assume the colored points on the real line are dynamic, i.e., they can be inserted or deleted. Our goal here is to maintain the minimal color-spanning intervals and, specifically, the smallest color-spanning interval. This is the main ingredient of our algorithm to compute the smallest color-spanning square for a set of colored points in the plane.

In the previous section, we showed the minimal color-spanning intervals for a set of colored points on the real line can be considered as the maximal points in the transformed plane. Therefore, maintaining the minimal color-spanning intervals reduces to maintaining the skyline points in the transformed plane. Overmars and van Leeuwen [11] proposed a data structure for maintaining the skyline. In the following we briefly go over their data structure.

Let $\mathcal{Q}$ be a set of n points in the plane. Overmars and van Leeuwen [11] used a binary search tree T which stores all points in its leaves in the sorted order by their y-coordinates. Moreover, an internal node v is augmented with a concatenable tree, e.g. a 2-3 tree, which stores the skyline of points in the subtree rooted at v that are not contained in the skyline of points in the subtree rooted at v's parent. The skyline of all points in $\mathcal{Q}$ is augmented in the root of T. In addition, the cut point where the skyline split is stored in v.

If a point p is inserted, a procedure $\mathit{Down}(T,p)$ goes down the tree to locate p and construct the skyline of points for subtrees of children of each internal node u on the path. Suppose $\mathit{Down}(T,p)$ is running and the skyline of u's subtree has been computed in u's parent in the previous step. Then, from the cut point of u's children, it is possible to split the u's skyline at that point and to merge split parts with the lists stored in augmented trees of u's children. After inserting point p to a leaf of tree T, the other procedure $\mathit{Up}(T,p)$ goes up the tree to the root of T and reconstruct the augmented trees and cut points for children of each internal node placed on the path. If a point is deleted, the procedures work similarly. These procedures split or merge two concatenable queues in $O(\log n)$ time in each internal node on the path from root to inserted or deleted leaf and totally needs $O(\log^2 n)$ time. In addition, they showed their data structure and algorithms use linear space [11].

We use the described structure to maintain the smallest color-spanning interval which is the maximal point with the minimum distance to the line $y = x$ (the minimum maximal point) in the transformed plane. Furthermore, we define some pointers to maintain the minimum maximal point per insertion and deletion. For each internal node v in augmented trees, we set two pointers p^*_{m1} and p^*_{m2} to the leaves corresponding to the consecutive skyline points which together define the minimum maximal point in the subtree rooted at v. In fact, p^*_{m1} and p^*_{m2} indicate the endpoints of the smallest color-spanning interval for the points in the subtree rooted at v. Moreover, let p^*_l and p^*_r be pointers respectively to the leftmost and the rightmost leaves in the subtree rooted at v. We use the above pointers to update the data structure in $O(\log^2 n)$ time and maintaining the minimum maximal point. We omit the details due to the space constraint.

Theorem 2. *For a given set of n points with k colors on the real line, the smallest color-spanning interval can be maintained in $O(\log^2 n)$ update time per insertion and deletion and $O(n)$ space.*

Proof. When a new point p with color c is inserted between the points $p_{c_{i-1}}$ and p_{c_i} in the c-colored list, the point $p^*_{c_i} = (p_{c_i}, p_{c_{i-1}})$ should be deleted from the transformed plane and two new points $p^*_{c_i} = (p_{c_i}, p)$ and $p^* = (p, p_{c_{i-1}})$ are inserted. Deletion of a point similarly needs a constant number of insertions and deletions in the transformed plane. □

We showed how we can maintain the smallest color-spanning interval for dynamic points on the real line. This is an important tool which helps us in the next section to compute the smallest color-spanning square for a given set of colored points in the plane.

3 The Smallest Color-Spanning Square

In this section, we focus on computing the smallest area/perimeter color-spanning axis-parallel square. Suppose we are given a set of n points with k colors in the plane. The smallest color-spanning square is an axis-parallel square which contains all colors and its area/perimeter is minimum.

Recall that this problem can be solved using the upper envelope of Voronoi surfaces [8] in $O(kn \log n)$ time. In fact, all the previously used methods for computing the smallest color-spanning objects such as rectangle, circle, etc. generally test all the minimal objects. We show that there are $\Omega(kn)$ minimal color-spanning squares in the worst case. This indicates that any algorithm testing all minimal color-spanning squares runs in $\Omega(kn)$ time. Next, we present an algorithm running in $O(n \log^2 n)$ time that computes the smallest color-spanning square without testing all minimal color-spanning squares.

We first explain how a minimal color-spanning square can be represented. As illustrated in Figure 4, a minimal color-spanning square is defined with two, three or four points with different colors on its edges under the assumption that point coordinates are different (recall that this assumption is just for the ease of the presentation). In all cases the minimal color-spanning square is bounded by two points on opposite sides. So, it suffices to consider the minimal color-spanning squares of case 1 instead of all. Moreover, we are interested in counting the minimal color-spanning squares with different contained points. We first give a lower bound for the number of minimal color-spanning squares.

Lemma 2. *There is a configuration of n points with k colors in the plane in which there are $\Omega(kn)$ minimal color-spanning squares.*

Proof. Let's place a set of $2k$ points with k colors in the configuration shown in Figure 5. In this pattern, we have to fix the left and the right edges in a way that they span the half of the colors (1 to $\frac{k}{2}$). So, there are $\frac{k}{2} + 1$ choices for fixing the right and the left edges. Similarly, for a pair of fixed left and right edges, it is possible to obtain $\frac{k}{2} + 1$ pairs to be the top and the bottom sides. Therefore, there are $\Omega(k^2)$ minimal color-spanning squares in the pattern of Figure 5 using $2k$ points. By repeating this pattern $\frac{n}{2k}$ times, we achieve $\Omega(kn)$ minimal color-spanning squares for a set of n points with k colors. □

Now, we start describing the steps of our main algorithm. Suppose the points are sorted in a descending order according to their y coordinates. The algorithm

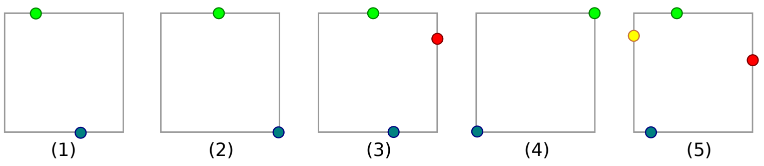


Fig. 4. The cases of a minimal color-spanning square

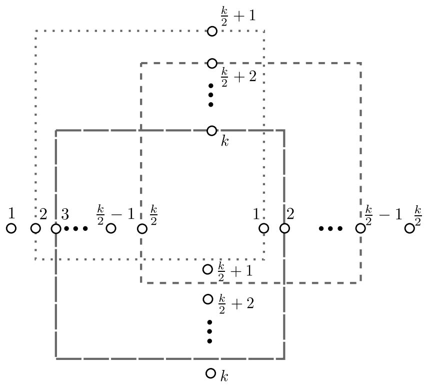


Fig. 5. $\Omega(k^2)$ minimal color-spanning squares with $2k$ points

sweeps the points with two lines L_b and L_t from top to bottom and variable d denotes the vertical distance of the sweep lines—see Figure 6 for more illustration. In fact, the lines L_b and L_t, respectively, define the bottom and the top edges of the desired square. At the beginning, L_b and L_t pass through the topmost point. We move down the sweep lines as follows:

– Move down the line L_t to the next point if there exist at least one color-spanning square in the strip bounded by L_b and L_t—see Figure 6(a).
– Move down the line L_b to the next point if there is no color-spanning square in the strip bounded by L_b and L_t—see Figure 6(b).

Now, it remains to show how we can find out if there is a minimal-color-spanning square in the strip of lines L_b and L_t. Suppose the points in the strip are projected onto the real line; let $\mathcal{P}$ be the union of this set of projected points and the additional points from each color at infinity as defined in the previous section. The following simple but important observation describes the necessary and sufficient conditions that a minimal color-spanning square exists inside the strip bounded by L_b and L_t.

Observation 1. *There is a minimal color-spanning square in the strip of lines L_b and L_t if and only if for the points in $\mathcal{P}$ the length of the smallest color-spanning interval is at most d.*

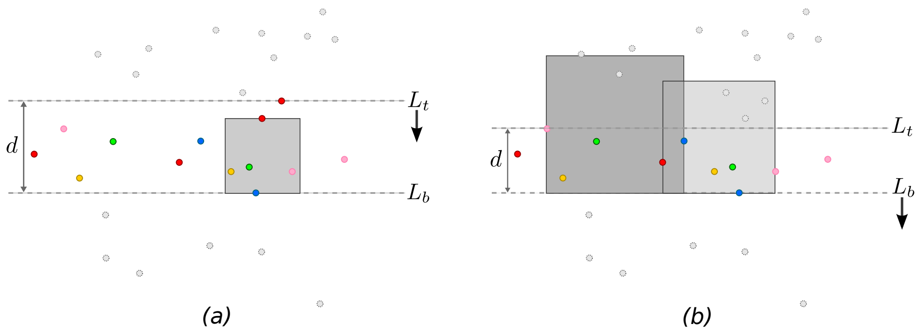


Fig. 6. Sweeping the points with two lines L_b and L_t

According to this observation, the algorithm only considers the projected points $\mathcal{P}$. Indeed, when L_b reaches a new point or a point goes out of the strip when L_t moves down, we insert or respectively delete a point from the dynamic structure of maintaining the smallest color-spanning interval that we described in the previous section. In addition, while L_t is moving down, at the time the smallest color-spanning interval becomes greater than d, the solution should be updated. We give the following theorem.

Theorem 3. *For a given set of n points with k colors in the plane, the smallest color-spanning axis-parallel square can be computed in $O(n \log^2 n)$ time and $O(n)$ space.*

Proof. Let s be the smallest color-spanning square which its top and bottom sides are defined with points p_t and p_b respectively. Suppose t_1 is the time L_b reaches p_b and t_2 is the time L_t stops at point p_t. There are two possibilities here. The case $t_1 < t_2$ means L_b is at point p_b while L_t is above the point p_t. Since there is at least one color-spanning square between the sweep lines, L_t moves down until it reaches point p_t. When L_t leaves out p_t, the length of the smallest color-spanning interval becomes greater than d and s has been visited. Otherwise s is not the solution which is a contradiction. Similarly, if $t_1 > t_2$, L_t stops at point p_t while L_b moves down until it reaches to the point p_b. Since s is the solution, L_b must stop at point p_b. Therefore, we visit the smallest color-spanning square in both cases. To analyse the running time of our algorithm, we noted that each point p enters the strip and is eliminated from it once when L_b and respectively L_t reach it. Therefore, in total we have $O(n)$ insertions and deletions and by Theorem 2 each operation spends $O(\log^2 n)$ time to maintain the smallest color-spanning interval for the points in $\mathcal{P}$. So, the algorithm runs in $O(n \log^2 n)$ time and $O(n)$ space. □

4 Conclusion

For a set of colored points in the plane, the problem of computing the smallest color-spanning axis-parallel square was studied in this paper. The previous methods for the smallest color-spanning objects test all minimal candidates. We showed although there could be $\Omega(kn)$ minimal color-spanning squares, there exists an algorithm that compute the smallest color-spanning square without testing all minimal ones. As the main ingredient of our algorithm, we first presented a dynamic data structure to maintain the smallest color-spanning interval for dynamic points on the real line. Then we used this dynamic structure to run our algorithm in $O(n \log^2 n)$ time and $O(n)$ space for computing the smallest color-spanning axis-parallel square. Our algorithm is independent from number of colors and faster than the algorithm mentioned by Abellanas et al.[1] $\big($running in $O(kn \log n)$ time$\big)$ which is based on computing the upper envelope of Voronoi surfaces [8].

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: Smallest Color-Spanning Objects. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 278–289. Springer, Heidelberg (2001)
2. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: The Farthest Color Voronoi Diagram and Related Problems. Tech. Report. University of Bonn (2006)
3. Consuegra, M.E., Narasimhan, G., Tanigawa, S.: Geometric Avatar Problems. Tech. Report TR-2013-02-25. Florida International University (2013)
4. Das, S., Goswami, P.P., Nandy, S.C.: Smallest Color-Spanning Object Revisited. Int. J. Comput. Geometry Appl. 19, 457–478 (2009)
5. Fan, C., Ju, W., Luo, J., Zhu, B.: On Some Geometric Problems of Color-Spanning Sets. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 113–124. Springer, Heidelberg (2011)
6. Fan, C., Luo, J., Zhong, F., Zhu, B.: Expected Computations on Color Spanning Sets. In: Fellows, M., Tan, X., Zhu, B. (eds.) FAW-AAIM 2013. LNCS, vol. 7924, pp. 130–141. Springer, Heidelberg (2013)
7. Gupta, P., Janardan, R., Smid, M.H.M.: Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. J. Algorithms 19(2), 282–317 (1995)
8. Huttenlocher, D.P., Kedem, K., Sharir, M.: The Upper Envelope of Voronoi Surfaces and Its Applications. Discrete Computational Geometry 9, 267–291 (1993)
9. Löffler, M.: Data Imprecision in Computational Geometry. Ph.D. Thesis. Utrecht University (2009)
10. Matoušek, J.: On Enclosing k Points by a Circle. Information Processing Letters 53(4), 217–221 (1995)
11. Overmars, M.H., van Leeuwen, J.: Maintenance of Configurations in the Plane. J. Comput. Syst. Sci. 23, 166–204 (1981)
12. Smid, M.H.M.: Finding k Points With a Smallest Enclosing Square. MPI-I-92-152, Max-Planck-Institut Inform., Saarbrücken, Germany (1992)

Euclidean Traveling Salesman Tours through Stochastic Neighborhoods

Pegah Kamousi and Subhash Suri

Department of Computer Science, UC Santa Barbara, CA 93106

Abstract. We consider the problem of planning a shortest tour through a collection of neighborhoods in the plane, where each neighborhood is a disk whose radius is an *i.i.d.* random variable drawn from a known probability distribution. This is a *stochastic* version of the classic traveling salesman problem with neighborhoods (TSPN). Planning such tours under uncertainty, a fundamental problem in its own right, is motivated by a number of applications including the following data gathering problem in sensor networks: a robotic *data mule* needs to collect data from n geographically distributed wireless sensor nodes whose communication range r is a random variable influenced by environmental factors.

We propose a polynomial-time algorithm that achieves a factor $O(\log\log n)$ approximation of the *expected length of an optimal tour*. In data mule applications, the problem has an additional complexity: the radii of the disks are only *revealed* when the robot reaches the disk boundary (transmission success). For this *online* version of the stochastic TSPN, we achieve an approximation ratio of $O(\log n)$. In the special case, where the disks with their *mean radii* are disjoint, we achieve an $O(1)$ approximation even for the online case.

1 Introduction

Planning under uncertainty is a central problem in many domains. In this paper, we consider a variant of the classical TSP problem under a stochastic scenario. Our setting requires planning an optimal tour that visits each of the n regions in the plane, called *neighborhoods*, under the Euclidean metric. The regions in our problem are disks $D_i = (c_i, r_i)$, where c_i is the (fixed) center and r_i denotes the (random) radius of disk D_i. The disk radii are random variables drawn independently and identically from some probability distribution, and so a random instance of the problem involves an arbitrary set of disks, with varying radii and an arbitrary overlap pattern. Our problem is to minimize the *expected* length of the tour visiting these disks; the problem is clearly NP-hard because it subsumes the classical Euclidean TSP by setting the mean and the variance of the probability distribution to zero.

The TSP with stochastic neighborhoods is motivated by natural applications where the target sites can be “visited” from afar—for instance, inspecting an asset or transferring data over wireless channels. One can imagine that a “visibility-based” monitoring of a set of distributed assets leads to a stochastic neighborhood problem since many unpredictable factors may influence the "lighting", changing the range of visual inspection. In distributed sensor networks, the use of "robotic data mules" is growing in acceptance due to energy constraints and the difficulty of transporting data over multiple hops [5,28]. However, the wireless range of radio transceivers exhibits significant fluctuations and randomness [24], which naturally leads to a stochastic version of the TSP with connected neighborhoods. Indeed, these type of applications entail another source of complexity: the precise value of the disk radius (communication range) is only revealed when the tour reaches the site. Thus, the problem involves both the stochastic and the *online* element. In this paper, we will consider both the offline and the online versions of the TSP with stochastic neighborhood. We begin with some notation and an informal definition of the problem.

Let $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ be a set of n random disks in the plane, where each disk $D_i = (c_i, r_i)$ has a fixed center c_i, but its radius r_i is a random variable drawn independently and identically from a probability distribution with mean μ. The probability distribution can be arbitrary subject only to the following weak constraints: (1) its domain is the positive reals, (2) it attains its maximum at μ and decays monotonically on either side of the mean, and (3) the probability of observing a radius r decreases quickly as r goes from μ to 0. In particular, if $F(x)$ is the cumulative probability function, then we require that $F(\mu/\alpha) \leq O(e^{-\alpha})$. (See Section 2 for more details on the distribution.) Given such a collection of disks, let L^* be the length of an optimal tour of $\mathcal{D}$, which is a random variable, and let $\mathbb{E}[L^*]$ be the expected value of this random variable over all realizations of the disk neighborhoods $\mathcal{D}$. In the offline case, we assume that the algorithm knows the input instance at the start of the tour, while in the online case the radii of the disks are revealed only when the tour reaches each disk. We prove the following three results in this paper:

1. We can compute a TSP tour through n stochastic disks whose expected length is within factor $O(\log \log n)$ of $\mathbb{E}[L^*]$ in polynomial time.
2. If the radii of the stochastic disks are revealed online, our algorithm achieves an $O(\log n)$ approximation of $\mathbb{E}[L^*]$.
3. If the disks are disjoint when they all appear with radius μ, then the approximation factor improves to $O(1)$ in both offline and online cases.

Related Work

There is a long history of research on probabilistic or stochastic traveling salesman problems. For instance, the celebrated result of Bearwood et al. [3] shows that (in the limit) the optimal TSP through n *i.i.d.* random points in $[0,1]^2$ has length $\Theta(\sqrt{n})$. Bertsimas and Jaillet [4,19] consider a setting where each point in a given set has an (independent) *activation* probability. They compute a single *a priori* tour, and on any random instance the tour is simply short-cut, visiting only the active points. Their objective is to find the a priori tour minimizing the expected cost over all random instances. Recent work on the *a priori* TSP and a related *universal* TSP includes [14,17,25,26]. Another interesting thread includes *2-stage* stochastic optimization [16,27], where a part of the input (partial distribution) is revealed in the first stage, when the resources can be acquired more cheaply; the rest of the input is revealed in the second stage, when the resources are more expensive. The goal is to optimize the expected cost of building a network structure [8,13,18,20].

The research most relevant to our work concerns the TSP problem with neighborhoods (TSPN), first introduced by Arkin and Hassin [1]. The problem is known to be APX-hard when the neighborhoods are general overlapping polygons [7,15], and hence the approximation algorithms have focused on either disjoint or "fat" neighborhoods. In particular, if the regions are disjoint disks of identical size, then there exists a PTAS [9]. If the regions are disjoint, fat polygons of comparable size, there also exists a PTAS [12]. Other results include a quasipolynomial-time approximation scheme (QPTAS), in any fixed dimension, when the regions are fat and disjoint [6]; an $O(1)$-approximation for disjoint, convex, and fat regions of arbitrary diameters [7], a PTAS under the assumption of bounded overlap [22], and an $O(1)$-approximation for disjoint neighborhoods of any size and shape [23]. Without the assumption of disjointness, the approximation results have tended to assume *regions with comparable diameters*. In particular, the best results include a constant factor approximation when the regions are connected polygons [9], convex and fat [10,11], or it is required to visit each neighborhood at one of a finite subset of points.

When the regions are neither disjoint nor of roughly the same size, the best approximation ratio known is $O(\log n)$ [10,21]. In our setting, the stochastic disks can have arbitrarily large radii and overlap in arbitrary ways, and so the prior work does not give an approximation ratio better than $O(\log n)$. When the radii are revealed *online*, no prior work seems to be known. In our stochastic setting, instead of comparing the performance of the algorithm for every single realization, we are interested in the *expected* performance over all the realizations.

2 Technical Preliminaries for the Stochastic TSP

Let $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ be a set of n random disks in the plane, where each disk $D_i = (c_i, r_i)$ has a fixed center c_i and a random radius r_i drawn from a probability distribution ϕ with mean μ (we highlight that the disk radii are identically distributed). Our analysis relies on a few assumptions about ϕ. In particular, we assume that (1) the domain of ϕ is the positive reals, and (2) ϕ attains its maximum at μ and then decays monotonically on either side of the mean. Finally, a reasonable probability distribution for the radius must be *scale invariant*: the probability of observing r should depend only on the *ratio* μ/r, independent of the distance scale. Thus, instead of a bound on the variance of ϕ, we assume that the cumulative probability function satisfies $F(\mu/\alpha) \leq O(e^{-\alpha})$, for $\alpha > 1$. In other words, we require that the ratio μ/r follow a light-tailed distribution [2]: Normal, exponential, and many other natural distribution are light-tailed. We do not require the distribution to be symmetric, and the radii can assume arbitrarily large values above the mean μ. (The assumption of a light-tailed distribution also conforms with the empirical observation of the transmission range in wireless sensors, where the probability of transmission failure drops quickly within the *reference distance* from the sensor [24].)

With the disk centers fixed, we may view the set $\mathcal{D}$ as an n-dimensional random vector $\mathcal{R} = (r_1, r_2, \ldots, r_n)$. Let $\mathcal{I}_\mathcal{R}$ denote the set of all the possible instances (realizations) of the vector $\mathcal{R}$. Each $I \in \mathcal{I}_\mathcal{R}$ uniquely identifies a particular instance of our TSP with neighborhoods. The probability distribution of $\mathcal{R}$, denoted $\phi_\mathcal{R}$, can be obtained from the marginal distributions of the radii. That is, for an instance $I = (x_1, x_2, \ldots, x_n)$, we have $\phi_\mathcal{R}(I) = \phi_\mathcal{R}(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \phi(x_i)$, where $\int_{x_1} \cdots \int_{x_n} \phi(x_1) \cdots \phi(x_n)\, dx_1 \cdots dx_n = 1$.

The expected value of $\mathcal{R}$ is the vector $\mu^{(n)}$, where each of the n disks has the radius equal to the mean value μ. This particular instance plays an important enough role in our analysis that we reserve a special symbol M for it. The optimal TSPN tour for the instance M is called $\textsc{Opt}(M)$.

Let the random variable L^* measure the length of the shortest tour over the sample space $\mathcal{D}$. The expected value of L^* can be computed as follows:

$$\mathbb{E}[L^*] = \int_0^\infty \cdots \int_0^\infty L^*(x_1, \ldots, x_n) \cdot \phi_\mathcal{R}(x_1, \ldots, x_n)\, dx_1 \ldots dx_n,$$

where $L^*(I)$ denotes the value of L^* for instance I.

Given any polygonal path or cycle T, we use $|T|$ to denote its Euclidean length, i.e., the sum of the lengths of its segments. To simplify our presentation, we also assume a fixed start point s_0 for the tour that lies at least 2μ away from all the disk centers. This technical assumption, which does not affect the general validity of our results, helps us ignore some special cases, such as when all disks have a common intersection in the instance M, causing $|T(M)| = 0$.

2.1 Bounding the Expected Optimal

We begin with a theorem establishing the importance of the instance M, where all disks occur with mean radii. It basically shows that *optimal of the mean is a good lower bound on the mean of the optimal.* Due to the page limit, this along with other proofs are omitted in this extended abstract.

Theorem 1. $|\textsc{Opt}(M)| \leq 2\,\mathbb{E}[L^*]$.

2.2 The High Level Strategy and a Partial Order of Disks

All of our stochastic TSP algorithms employ the following three-step strategy: first, we compute an $O(1)$ approximation $T(M)$ for the mean-radius instance M—that is, $|T(M)| = O(|\textsc{Opt}(M)|)$; second, we subdivide $T(M)$ into several *blocks* and assign a subset of the disks to each block; finally, for a random instance I, we construct a tour by visiting disks in the block order given by $T(M)$. (We note that following the same path as $T(M)$ does not necessarily visit all the disks, since their radii in I could be smaller than the mean value. So, the block order is just a high-level clue about how "subsets" of disks should be visited.)

In the rest of this section, we describe the first (and simplest) of these three steps, while the other steps are the focus of next section. Our algorithm for approximating the TSP for instance M is based on some classical ideas for approximating the TSP of disks. In particular, if the neighborhoods are convex regions of equal diameter in the plane, then a polynomial algorithm is known for a constant factor approximation of the TSP visiting all the neighborhoods [1,9]. The approximation algorithm works by choosing a *representative point* in each convex region and finding an almost optimal tour of these points.

In this spirit, consider the instance M, which has n (possibly intersecting) disks, each of radius μ. We call a set of vertical lines a *line cover*, if each disk is intersected by at least one of these lines. A line cover with the minimum number of lines is easily computed by a simple greedy scan: the first line is chosen to pass through the rightmost point of the leftmost disk; remove all the disks intersected by this line, and repeat until all the disks are covered. We can make two simple observations: first, each disk is intersected by precisely one line in the cover, which we call the *covering line* of this disk; and second, two adjacent lines of the cover are at least 2μ apart. See Figure 1(a). For each disk D_i, the point where the covering line of D_i meets its horizontal diameter is selected as its unique *representative point*. Following the algorithm of [1,9], we then compute a $(1+\epsilon)$-approximate tour of these representative points. Call this tour $T(M)$. Then, by Theorem 1, we have the important result that $T(M) = O(\mathbb{E}[L^*])$.

Unfortunately, by itself, $T(M)$ is not a good tour for a random instance I—in fact, it may not even visit some of the disks whose radius in I is smaller than μ. However, we show that it provides a good high-level clue about the rough order in which to visit the disks in *any random instance*. Let us fix an orientation of $T(M)$, say clockwise, and let $A = \{a_1, \ldots, a_n\}$ denote the sequence of representative points of disks in M in the order they are visited by $T(M)$, starting with the first disk visited following the initial point s_0. Recall that all the n representatives lie on the covering lines, which have a minimum separation of 2μ. We now partition the sequence A into *chunks* of consecutive points $A_1, A_2, \ldots, A_m$, such that each chunk contains points that belong to the same covering line *and* are consecutive along the tour $T(M)$. W.l.o.g., let A_0 consist of the singleton initial point s_0. We note that the representative points of a covering line may be partitioned into more than one such chunk. See Figure 1(b) for an example.

Let ℓ_i, for $i = 1, 2, \ldots, m$, be the line segment joining the *lowest* and the *highest* (by y-coordinate) point in A_i. Clearly, ℓ_i covers all the points in A_i, and thus visits the mean radius disks associated with them. We will use these chunks A_i to divide $T(M)$ into m blocks $B_1, \ldots, B_m$, where B_i is the portion of $T(M)$ visiting the points in A_i together with the line segment connecting the last point in A_{i-1} to the first point in A_i. That is, B_i is the part of $T(M)$ starting after its last contact with ℓ_{i-1} and ending right after its last contact with ℓ_i. See Figure 1(c). Since the minimum distance between ℓ_i and ℓ_{i+1} is 2μ, we can lower bound the length of the B_i by $2\mu + |\ell_i|$; the initial block $|B_1|$, being an exception,

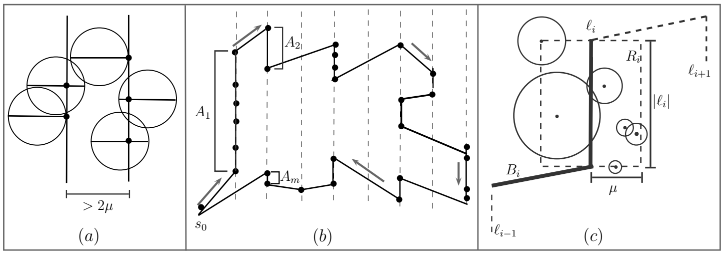


Fig. 1. (a) A set of covering lines; (b) a possible structure of $T(M)$ (c) a block B_i and rectangle R_i

is lower bounded by $\mu + |\ell_1|$, since s_0 is at least μ away from its closest disk. From Theorem 1 and the fact that $|T(M)| = \sum_{i=1}^{m} |B_i|$, we have

Observation 1. $\sum_{i=1}^{m} |B_i| = O(\mathbb{E}[L^*])$.

We say that all the disks covered by ℓ_i are *assigned* to the block B_i, and these blocks form the desired *partial order* on our input disks: all disks assigned to block i precede any disk assigned to block j if $i < j$. By construction, the centers of all the disks assigned to block B_i lie within the rectangle R_i of dimensions $|\ell_i| \times 2\mu$, with vertical axis ℓ_i (see Figure 1(c)). We will argue that for any random instance, by visiting all the disks of each block in the partial order imposed by blocks, we obtain a tour that achieves a $O(\log\log n)$ factor approximation of the expected optimum. Before discussing the strategy to visit the disks in each block, we note a simple geometric property of the optimal disk tour. The proof is simple: the optimal tour must be polygonal, has at most one vertex per disk, and cannot self-intersect—otherwise it can be shortcut, violating optimality.

Lemma 1. *The optimal TSPN tour of any instance I is a polygonal cycle with at most n vertices that does not self-intersect.*

Suppose Opt is an optimal tour of n disks in the plane, and consider an axis-aligned square W, called a *window*, entirely inside the minimum bounding box of the disks. Focus on tour fragment $P = \text{Opt} \cap W$, namely, the portion of Opt contained in W, which may be composed of multiple disconnected pieces, as shown in Figure 2. Then P must visit all the disks completely contained in W. The following lemma shows that P together with the boundary of W is lower bounded by the shortest tour that visits all the disks contained in W, up to a constant. In particular, suppose Opt$'$ is the optimal tour for the subset of disks contained completely inside W.

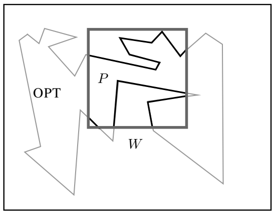


Fig. 2. The portion of Opt contained in W

Lemma 2. $|\text{Opt}'| \leq 2(|P| + |W|)$, *where $|W|$ is the perimeter of W .*

Our discussion so far applies to the general stochastic TSPN problem: computing the approximately optimal tour for the mean instance M, the partial ordering of disks and the block partition all only require knowledge of the mean radius and the disk centers. However, the last key step that computes a good approximation tour for each block B_i crucially depends on whether we know the radii of the random instance beforehand or not. Therefore, the following discussion now separately considers the *offline* and the *online* versions of the problem: in the former, the radii of the random instances are known to the algorithm at the beginning, while in the latter the algorithm only learns the radius of a disk D_i when the tour reaches the boundary of D_i.

3 Stochastic Offline Tour

In this section, we describe an algorithm for visiting the stochastic disks in the offline setting: the salesman knows the disk radii of the given instance before starting the tour. We show how to construct a tour whose expected length is within factor $O(\log \log n)$ of the expected optimal.

In light of the discussion of the previous section, we only need to focus on constructing approximately optimal tours for each block B_i, for $i = 1, \ldots, m$, because their concatenation leads to an overall tour with length close to $\mathbb{E}\left[L^* \right]$. We first recall that the centers of all the disks assigned to B_i lie within the (closed) rectangle R_i with dimensions $|\ell_i| \times 2\mu$, and centered at the midpoint of ℓ_i. We partition R_i into $2\mu \times 2\mu$ squares; (the last "square" may be a rectangle of width 2μ and height smaller than 2μ). We construct the tour separately for each of these squares, visiting the disks *whose centers lie in the square.* The concatenation of these subtours gives the final tour. With this preamble, the next subsection considers the following key problem: given n disks whose centers lie inside a square of side length 2μ, construct a tour visiting them. We then explain and analyze the algorithm to combine these subtours in subsection 3.2.

3.1 Constructing a Subtour within a Square

Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a random instance of the stochastic TSPN problem where the centers of all the disks lie inside a square R of dimensions $2\mu \times 2\mu$, where μ is the mean radius of the disks. We show how to construct a tour visiting these disks with expected length $O(\log \log n)$ times the expected optimal. We begin with an idea used in the work of Elbassioni et. al. [10] for the deterministic TSPN problem on intersecting neighborhoods.

First, let $\mathcal{D}_{\text{in}} \subseteq \mathcal{D}$ denote the set of disks contained in the interior of R. If $\mathcal{D}_{\text{in}} = \emptyset$, then the boundary of R visits all the disks, and this is an easy case. Otherwise, $\mathcal{D}_{\text{in}} \neq \emptyset$, and we proceed as follows. We let $N_2(D_i) \subseteq \mathcal{D}$ denote the *2-neighborhood* of disk $D_i = (c_i, r_i)$, which is the set of disks in $\mathcal{D}$ within distance $2r_i$ of c_i. That is, $N_2(D_i)$ is the set of disks that intersect a disk of radius $2r_i$ centered at c_i. We call the disk D_i the *core* of $N_2(D_i)$. We use the 2-neighborhoods to form a disjoint cover of $\mathcal{D}_{\text{in}}$, by the following iterative algorithm.

Initially, $\mathcal{N} = \emptyset$. Choose the disk $D_i \in \mathcal{D}$ with the smallest radius, and add the 2-neighborhood $N_2(D_i)$ to $\mathcal{N}$, with D_i as its *core*. Remove all the disks of $N_2(D_i)$ from $\mathcal{D}$, and iterate until $\mathcal{D}$ is empty. Clearly, each disk $D_j \in \mathcal{D}_{\text{in}}$ is assigned to $\mathcal{N}$ at some point, and we identify it with the core disk D_i whose 2-neighborhood added D_j to $\mathcal{N}$. Without loss of generality, suppose $D_1, D_2, \ldots, D_k$ are the core disks selected by the covering algorithm in this order. Clearly, by the disk selection rule, any two core disks are disjoint, that is, $D_i \cap D_j = \emptyset$, for $1 \le i, j \le k$, and the radii are in increasing order, namely, $r_1 \le r_2 \le \cdots \le r_k$.

Lemma 3. *Let $N'(D_i) \subseteq N_2(D_i)$ be the set of disks added to $\mathcal{N}$ when D_i is chosen as core. Then there is a tour of length at most $O(r_i)$ visiting all the disks of $N'(D_i)$.*

Let OPT' denote an optimal tour that visits all the disks of $\mathcal{D}_{\text{in}}$. The following key lemma gives a lower bound on $|\text{OPT}'|$ for *any* instance of the problem in terms of just the radii of core disks. An analogue of this Lemma can also be found in [10] (and also in [22], in a slightly more general form).

Lemma 4. *Let $\{D_1, \ldots, D_k\}$, for $k \ge 2$, be the set of core disks whose 2-neighborhoods form the disjoint partition of $\mathcal{D}_{in}$. Then, $|\text{OPT}'| \ge \sum_{i=1}^{k-1} \left(\frac{r_i}{\lceil \log k \rceil} \right)$.*

Remark: The lower bound of the preceding lemma is tight in the worst-case. We can construct a set of core disks for which the optimal tour is *at most* $1/\log k$ times the sum of radii. In [22] a similar lower bound is presented for fat regions.

Lemma 5. *The number of disks selected as a core in a disjoint partition of $\mathcal{D}_{in}$ whose radius exceeds $\mu/\log n$ is at most $O(\log^2 n)$.*

Lemma 6. *In any random instance $I \in \mathcal{I}_{\mathcal{R}}$, the expected number of disks $D_i \in \mathcal{D}_{in}$ with radius smaller than $\mu/\log n$ is a constant.*

The next theorem shows how to construct a tour of $\mathcal{D}$ using the tour of $\mathcal{D}_{\text{in}}$. Please see the appendix for the proof.

Theorem 2. *In polynomial time, we can compute a tour $T(\mathcal{D})$ visiting all the disks of $\mathcal{D}$ such that $\mathbb{E}[|T(\mathcal{D})|] \le \mu + O(\log\log n \; \mathbb{E}[|\text{OPT}'|])$, where OPT' denotes the optimal tour on $\mathcal{D}_{in}$.*

3.2 Combining the Subtours

We now stitch together these subtours spanning the disks whose centers lie in $2\mu \times 2\mu$ size squares to construct the final tour. See Figure 3 for illustration. In particular, suppose $S_i = \{R_{i1}, R_{i2}, \ldots\}$ is the partition of the rectangle R_i into these $2\mu \times 2\mu$ squares, and let T_{ij} be the $O(\log\log n)$-optimal tour (obtained using Theorem 2) for the disks whose centers lie in R_{ij}, where $R_{ij} \in S_i$. Let T_i be the path obtained by concatenating the tours T_{ij}, for $\{j : R_{ij} \in S_i\}$, where $i = 1, \ldots, m$, adding at most $O(|\ell_i| + \mu)$ to the length. Finally, combine the paths

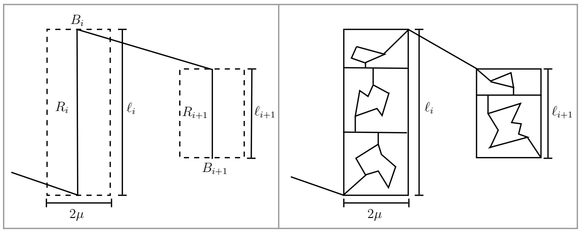


Fig. 3. Two blocks B_i and B_{i+1}, and the paths replacing them

T_i, for $i = 1, \dots, m$, to obtain a tour over $\mathcal{D}$, by connecting the boundary of R_i with the boundary of R_{i+1}. These connections add at most $\sum_{i=1}^{m} O(B_i)$ to the tour length. (Figure 3 illustrates this construction for blocks B_i and B_{i+1}.) It is easy to modify the resulting walk into a traveling salesman tour by doubling and shortcutting. Let $T(\mathcal{D})$ denote the resulting tour. The following theorem establishes the main result of this section.

Theorem 3. *In polynomial time, we can compute a tour $T(\mathcal{D})$ visiting the set $\mathcal{D}$ of stochastic disks, such that $\mathbb{E}\,[\,|T(\mathcal{D})|\,] \;=\; O(\log\log n)\cdot\mathbb{E}\,[\,L^*\,]$.*

4 Stochastic Online Tour

We now consider the *online* version of the TSP with stochastic disks, where the salesman learns the radius of each stochastic disk only on arriving at the boundary of the disk—in the data gather application, the disk radius is revealed when the robot is able to communicate with the sensor node. We propose an $O(\log n)$-approximation algorithm for the online version. In the special case where the mean radii disks are nearly disjoint, we achieve an $O(1)$-approximation, both for the online and the offline setting. (In practice, this is the more likely case.)

Our online algorithm also follows the same outline as the offline case, but uses a different (and simpler) scheme to visit all the disks inside each rectangle R_i. In particular, recall that the centers of the disks assigned to a block B_i lie inside or on the boundary of the rectangle R_i with dimensions $2\mu \times |\ell_i|$. We divide each R_i, for $i = 1, \dots, m$, into $\lceil \log n \cdot |\ell_i|/\mu \rceil$ horizontal strips of height (at most) $\mu/\log n$. See Figure 4(a). We now replace each segment ℓ_i of the mean radius tour $T(M)$ with a path that traverses the horizontal line segments of length 2μ in the middle of the strips one by one. Consider a disk $D_i = (c_i, r_i)$ whose center lies inside the current strip, and is not visited by the path so far. The tour *expects* to intersect that disk when it reaches the point with the same x-coordinate as c_i; if it fails to reach it, then it makes a detour towards c_i until it reaches the boundary of D_i and immediately returns to its position before the detour. We now analyze the expected length of this tour.

Let T_i be the path that replaces the block B_i. Figure 4(b) shows this path, which starts at the last point of B_{i-1} and ends at the first point of B_{i+1}, assuming an orientation of the tour $T(M)$.

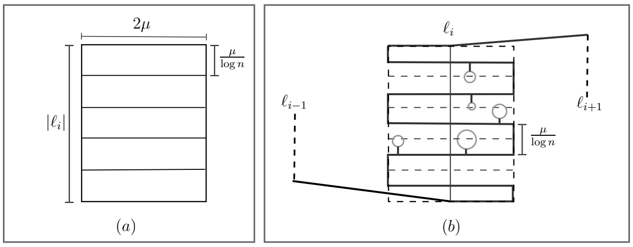


Fig. 4. (a) Partition of rectangle R_i into strips; (b) the path replacing B_i

Lemma 7. $\mathbb{E}\,[\,T_i\,] \;=\; O(\,\log n\,)\cdot|B_i|$.

Theorem 4. *If the radii of the set $\mathcal{D}$ of random disks are revealed online, we can compute in polynomial time a tour $T(\mathcal{D})$, where $\mathbb{E}\,[\,|T(\mathcal{D})|\,] = O(\log n)\cdot\mathbb{E}\,[\,L^*\,]$.*

Proof. Let $T(\mathcal{D})$ be the union of the paths T_i, for $i = 1,\ldots,m$, where T_i ends at the point where T_{i+1} begins. The tour T visits all the disks, and by Theorem 1 and Observation 1, we have the following, which completes the proof.

$$\mathbb{E}\,[\,|T(\mathcal{D})|\,] = \sum_{i=1}^{m}\mathbb{E}\,[\,|T_i|\,] = O(\log n)\cdot\sum_{i=1}^{m}|B_i| = O(\log n)\cdot|T(M)| = O(\log n)\cdot\mathbb{E}\,[\,L^*\,].$$

Almost Disjoint Mean Radius Disks. Finally, if the disks are not "too overlapping" in the instance M, we can obtain a simple $O(1)$-approximate tour of $\mathcal{D}$. We say that the set $\mathcal{D}$ has depth c if no point lies in the common intersection of more than c disks, as they appear in M. We note that *even with this assumption*, a random instance of the problem may still have arbitrarily large intersection depths. Nevertheless, we can prove the following result.

Theorem 5. *If the stochastic set $\mathcal{D}$ has a constant depth, then we can compute in polynomial time a tour $T(\mathcal{D})$ such that $\mathbb{E}\,[\,|T(\mathcal{D})|\,] \;=\; O(1)\cdot\mathbb{E}\,[\,L^*\,]$.*

References

1. Arkin, E.M., Hassin, R.: Approximation Algorithms For The Geometric Covering Salesman Problem. Discrete Applied Mathematics 55, 197–218 (1995)
2. Asmussen, S.: Applied Probability and Queues. Springer (2003)
3. Beardwood, J., Halton, J.H., Hammersley, J.M.: The Shortest Path Through Many Points. Proc. Cambridge Philos. Soc. 55, 299–327 (1959)
4. Bertsimas, D.: Probabilistic Combinatorial Optimization Problems. PhD thesis, Operation Research Center. MIT, Cambridge, MASS (1988)
5. Bhadauria, D., Tekdas, O., Isler, V.: Robotic Data Mules for Collecting Data over Sparse Sensor Fields. J. Field Robot. 28(3), 388–404 (2011)
6. Chan, T.-H.H., Elbassioni, K.: A QPTAS for TSP with Fat Weakly Disjoint Neighborhoods in Doubling Metrics. In: SODA 2010, pp. 256–267 (2010)
7. de Berg, M., Gudmundsson, J., Katz, M.J., Levcopoulos, C., Overmars, M.H., van der Stappen, A.F.: TSP with Neighborhoods of Varying Size. J. Algorithms 57(1), 22–36 (2005)

8. Dhamdhere, K., Ravi, R., Singh, M.: On Two-Stage Stochastic Minimum Spanning Trees. In: Jünger, M., Kaibel, V. (eds.) IPCO 2005. LNCS, vol. 3509, pp. 321–334. Springer, Heidelberg (2005)
9. Dumitrescu, A., Mitchel, J.S.B.: Approximation Algorithms for TSP with Neighborhoods in the Plane. Journal of Algorithms 48(1), 135–159 (2003)
10. Elbassioni, K.M., Fishkin, A.V., Sitters, R.: On Approximating the TSP with Intersecting Neighborhoods. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 213–222. Springer, Heidelberg (2006)
11. Elbassioni, K.M., Fishkin, A.V., Sitters, R.: Approximation Algorithms for the Euclidean Traveling Salesman Problem with Discrete and Continuous Neighborhoods. Int. J. Comput. Geometry Appl. 19(2), 173–193 (2009)
12. Feremans, C., Grigoriev, A.: Approximation Schemes for the Generalized Geometric Problems with Geographic Clustering, pp. 101–102 (2005)
13. Flaxman, A.D., Frieze, A., Krivelevich, M.: On the Random 2-Stage Minimum Spanning Tree. In: SODA 2005, pp. 919–926 (2005)
14. Gorodezky, I., Kleinberg, R., Shmoys, D., Spencer, G.: Improved Lower Bounds for the Universal and a priori TSP. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX 2010. LNCS, vol. 6302, pp. 178–191. Springer, Heidelberg (2010)
15. Gudmundsson, J., Levcopoulos, C.: Hardness Result for TSP with Neighborhoods. Technical report (2000)
16. Gupta, P., Martin, A., Ravi, R., Sinha, A.: Boosted Sampling: Approximation Algorithms for Stochastic Optimization. In: Proc. 36th Annual ACM Symposium on Theory of Computing, pp. 417–426 (2003)
17. Hajiaghayi, M.T., Kleinberg, R., Leighton, T.: Improved Lower and Upper Bounds for Universal TSP in Planar Metrics. In: SODA 2006, pp. 649–658 (2006)
18. Immorlica, N., Karger, M., Minkoff, D., Mirrokni, V.S.: On the Costs and Benefits of Procrastination: Approximation Algorithms for Stochastic Combinatorial Optimization Problems. In: SODA 2004, pp. 691–700 (2004)
19. Jaillet, P.: A priori Solution of a Traveling Salesman Problem in which a Random Subset of the Customers are Visited. Math. Oper. Res. 6(6) (1988)
20. Katriel, I., Kenyon-Mathieu, C., Upfal, E.: Commitment under Uncertainty: Two-Stage Stochastic Matching Problems. Theoretical Computer Science 408(2-3), 213–223 (2008)
21. Mata, C.S., Mitchell, J.S.B.: Approximation Algorithms for Geometric Tour and Network Design Problems. In: SoCG 1995, pp. 360–369. ACM (1995)
22. Mitchell, J.S.B.: A PTAS for TSP with Neighborhoods among Fat Regions in the Plane. In: SODA 2007, pp. 11–18 (2007)
23. Mitchell, J.S.B.: A Constant-Factor Approximation Algorithm for TSP with Pairwise-Disjoint Connected Neighborhoods in the Plane. In: SoCG 2010, pp. 183–191 (2010)
24. Muetze, T., Stuedi, P., Kuhn, F., Alonso, G.: Understanding Radio Irregularity in Wireless Networks. In: SECON 2008, pp. 82–90 (2008)
25. Platzman, L.K., Bartholdi III, J.: Spacefilling Curves and the Planar Travelling Salesman Problem. J. ACM 36(4), 719–737 (1989)
26. Shmoys, D.B., Talwar, K.: A Constant Approximation Algorithm for the *a priori* Traveling Salesman Problem. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 331–343. Springer, Heidelberg (2008)
27. Swamy, C., Shmoys, D.B.: Approximation Algorithms for 2-stage Stochastic Optimization Problems. SIGACT News 37(1), 33–46 (2006)
28. Tekdas, O., Isler, V., Lim, J.H., Terzis, A.: Using Mobile Robots to Harvest Data from Sensor Fields. Wireless Commun. 16(1), 22–28 (2009)

Detecting and Characterizing Small Dense Bipartite-Like Subgraphs by the Bipartiteness Ratio Measure*

Angsheng Li[1] and Pan Peng[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2] Department of Computer Science, Technische Universität Dortmund
angsheng@ios.ac.cn, pan.peng@tu-dortmund.de

Abstract. We study the problem of finding and characterizing subgraphs with small *bipartiteness ratio*. We give a bicriteria approximation algorithm `SwpDB` such that if there exists a subset S of volume at most k and bipartiteness ratio θ, then for any $0 < \epsilon < 1/2$, it finds a set S' of volume at most $2k^{1+\epsilon}$ and bipartiteness ratio at most $4\sqrt{\theta/\epsilon}$. By combining a truncation operation, we give a local algorithm `LocDB`, which has asymptotically the same approximation guarantee as the algorithm `SwpDB` on both the volume and bipartiteness ratio of the output set, and runs in time $O(\epsilon^2\theta^{-2}k^{1+\epsilon}\ln^3 k)$, independent of the size of the graph. Finally, we give a spectral characterization of the small dense bipartite-like subgraphs by using the kth *largest* eigenvalue of the Laplacian of the graph.

1 Introduction

We study the problem of finding subgraphs with small *bipartiteness ratio*. Let $G = (V, E)$ be an undirected graph. Let L, R be two disjoint vertex subsets and $U := L \cup R$. The bipartiteness ratio of L, R is defined as

$$\beta(L, R) = \frac{2e(L) + 2e(R) + e(U, \bar{U})}{\mathrm{vol}(U)}, \tag{1}$$

where $e(L), e(U, \bar{U})$ denote the number of edges in L and the number of edges leaving from U to the rest of the graph, respectively; and $\mathrm{vol}(U)$, called the volume of U, is defined to be the sum of degrees of vertices in U. The concept of bipartiteness ratio was recently introduced and used as a subroutine to designing

* The research is partially supported by NSFC distinguished young investigator award number 60325206, and its matching fund from the Hundred-Talent Program of the Chinese Academy of Sciences. Both authors are partially supported by the Grand Project "Network Algorithms and Digital Information" of the Institute of software, Chinese Academy of Sciences. The second author acknowledges the support of ERC grant No. 307696 and NSFC 61003030.

approximation algorithms for Max Cut problem by Trevisan [33]. In particular, Trevisan showed that this combinatorial object has close relation to the largest eigenvalue of the *Laplacian* of G, the Goemans-Williamson Relaxation and graph sparsification.

Another motivation of studying bipartiteness ratio is that it can be considered as a quality of *dense bipartite-like subgraphs*, which in turn characterize the communities (or clusters) in Web graphs [26,15]. More specifically, a dense bipartite-like subgraph is a pair of disjoint vertex subsets L, R such that 'most' of the edges involving the vertices in $L \cup R$ lie between L and R. Equivalently, we say that L, R form a dense bipartite subgraph if 'few' edges lie totally in L or R, or leaving $L \cup R$ to the rest of the graph. The latter formulation turns out to be well captured by the bipartiteness ratio measure of L, R, as in Definition (1), the numerator involves all the edges that are *not* between L and R, and the dominator involves all the edges incident to $L \cup R$. It is intuitive that the smaller the bipartiteness, the more likely it behaves like a dense bipartite subgraph. In real applications, we suggest first detecting sets with small bipartiteness ratio (using the algorithms below) and then combining some heuristic algorithms (such as the prune-filter technique used in [15]) to process the found sets and to better exploit the community structure of the Web graph.

Thus, we will use the bipartiteness ratio (abbreviated as *B-ratio*) as a measure of a set being dense bipartite-like. We want to extract subgraphs with small B-ratio, which corresponds to good cyber-communities. Furthermore, we are interested in finding *small* communities, which generally contains more interesting information than large communities and may be more substantial in large scale networks. For example, Leskovec et al. investigate the community structure of many real networks by the *conductance* measure [18,19], and they argue that large networks may have a core-periphery structure, where the periphery is composed of easily separable small communities and the nodes in the expander-like core are so intermingled that it is much harder to extract large communities (if exist) from it.

In order to make our algorithm practical, we would like to design a local algorithm to extract subgraphs with small B-ratio. A local algorithm, introduced by Spielman and Teng [29], is one that given as input a vertex, it only explores a small portion of the graph and finds a subgraph with good property, which has found applications in graph sparsificasion, solving linear equations [31], and designing near-linear time algorithms [32]. Local algorithms have also shown to be both effective and efficient on real network data (e.g, [18,20]).

1.1 Our Results

We give approximation, local algorithms and spectral characterization of finding the small subgraphs with small B-ratio, as we argued above, with the goal of extracting small cyber-communities. In the following, we will use the terminology of small dense bipartite-like subgraphs to indicate small subgraphs with small B-ratio.

We first give a bicriteria approximation algorithm for finding the small dense bipartite-like subgraph.

Theorem 1. *Assume that G has a set $U = (L, R)$ such that $\beta(L, R) \leq \theta$ and vol(U) $\leq k$, where $\theta < 1/4$ and $k > 4$, then for any $0 < \epsilon < 1/2$, there exists an algorithm* `SwpDB`(G, k, θ, ϵ) *that runs in polynomial time and finds a set (X, Y) such that vol($X \cup Y$) $\leq 2k^{1+\epsilon}$, and $\beta(X, Y) \leq 4\sqrt{\theta/\epsilon}$.*

Note that the approximation ratio does not depend on the size of the graph, since the algorithm is based on a spectral characterization of the B-ratio of the graph given by Trevisan [33] (see Lemma 1), which is analogous to the Cheeger's inequality for conductance (see more discussions below).

By incorporating a truncation operation we are able to give a *local algorithm* for the dense bipartite subgraphs.

Theorem 2. *If there is a subset $U = (L, R)$ of volume vol(U) $\leq k$ and B-ratio $\beta(L, R) \leq \theta$, where $\theta < 1/12$ and $k > 2560000$,then there exists a subgraph $U_\theta \subseteq U$ satisfying that vol(U_θ) $\geq$ vol(U)$/2$ and that if $v \in U_\theta$, then for any $0 < \epsilon < 1/2$, there exists a local algorithm* `LocDB`$(G, v, k, \theta, \epsilon)$ *finds a subgraph (X, Y) of volume $O(k^{1+\epsilon})$ and B-ratio $O(\sqrt{\theta/\epsilon})$. Furthermore, the running time of* `LocDB` *is $O(\epsilon^2\theta^{-2}k^{1+\epsilon}\ln^3 k)$.*

We remark that in both theorems, we can give alternative tradeoff on the bounds of parameters k and θ. For example, in Theorem 2, we can require $\theta < 0.03$ and $k > 11000$ instead (see the proof of the theorem).

Note that the local algorithm runs in time independent of the size of the graph and is sublinear time (in the size of the input graph, denoted as n) when the size of the optimal set is sufficiently smaller than n and the approximation ratio of the algorithm is almost optimal in that it almost matches the guarantee of Trevisan's spectral inequality for the B-ratio. This algorithm also improves the work of the second author [26], who gave a local algorithm for B-ratio guaranteeing that the output set has volume at most $O(k^2)$ and B-ratio at most $O(\sqrt{\theta})$.

Finally, as an application of the algorithm `SwpDB`, we give a spectral characterization of the small dense bipartite subgraph by relating the kth largest eigenvalue of the Laplacian of G to the B-ratio of some subsets with volume at most $O(2|E|/k^{1-\epsilon})$. More specifically, if we let $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$ be the eigenvalues of the Laplacian matrix $\mathcal{L}$ of the graph G, and define the *dense bipartite profile* of the graph as $\beta(k) := \min_{\substack{L,R: L\cap R=\emptyset \\ \mathrm{vol}(L\cup R)\leq k}} \beta(L, R)$, then our spectral characterization implies that

$$\beta(\mathrm{vol}(G)/k^{1-\epsilon}) \leq O(\sqrt{(2-\lambda_{n-k})\log_k n}). \tag{2}$$

1.2 Our Techniques

Our approximation algorithm is based on Trevisan's spectral characterization of the B-ratio $\beta(G)$ of the graph, which is the minimum B-ratio of all possible disjoint vertex subsets L, R, that is, $\beta(G) = \beta(\mathrm{vol}(G))$. Recall that $\lambda_0 \leq \lambda_1 \leq$

$\cdots \leq \lambda_{n-1}$ are the eigenvalues of $\mathcal{L}$. Instead of working directly on $\mathcal{L}$, we study a closely related matrix M, which we call the *quasi-Laplacian*, that has the same spectra as $\mathcal{L}$. Let $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{n-1}$ be the corresponding eigenvectors of M. Trevisan showed that if $\lambda_{n-1} \geq 2 - 2\theta$, then by a simple *sweep process* over the largest eigenvector $\mathbf{v}_{n-1}$, we can find a pair of subsets X, Y with B-ratio at most $2\sqrt{\theta}$. On the other hand, it is well known that the largest eigenvector $\mathbf{v}_{n-1}$ can be computed fast by the power method, which starts with a "good" vector $\mathbf{q}_0$ and iteratively multiplies it by M to obtain $\mathbf{q}_t$, and outputs $\mathbf{q}_T$ by choosing proper T. Hence, the power method combined with the sweep process can find a subset with B-ratio close to $\beta(G)$. However, such a method does not give a useful volume bound on the output set.

In order to find *small* dense bipartite subgraphs, we sweep each of the vector $\mathbf{q}_t$ and characterize $\mathbf{q}_t$ in terms of the minimum of B-ratio of all the small sweep sets (the sets found in the sweep process) encountered in all the T iterations. This is done by a *potential function* $J(\boldsymbol{p}, x)$, which has a nice convergence property that for general vector $\mathbf{p}$ and some x, $J(\mathbf{p}M, x)$ can be bounded by a function of $J(\mathbf{p}, x')$ and the B-ratio of the some sweep set (see Lemma 2). Using this property, we show inductively that if we choose $\mathbf{q}_0 = \chi_v$ for some vertex $v \in V$, $J(\mathbf{q}_t, x)$ can be upper bounded by a function in t, K and the minimum B-ratio of all the sweep sets of volume at most K for all $t \leq T$ (see Lemma 3). On the other hand, if the graph contains a small dense bipartite subgraph L, R of volume at most k, we prove that the potential function also increases quickly in terms of t and $\beta(L, R)$ (see Lemma 4), which will lead to the conclusion that at least one of the sweep set with volume at most K has B-ratio "close" to $\beta(L, R)$ by choosing proper K in terms of k and the starting vertex v.

To give local algorithms that run in time independent of the size of the graph, we need to keep the support size of the vectors $\mathbf{q}_t$ small in each iteration. This is done by a truncation operation of a vector that only keeps the elements with large absolute vector value. Let $\tilde{\mathbf{q}}_0 = \chi_v$ and iteratively define $\tilde{\mathbf{q}}_t$ to be the truncation vector of $\tilde{\mathbf{q}}_{t-1}M$. We show that both upper bound and lower bound on $J(\mathbf{q}_t, x)$ still approximately holds for $J(\tilde{\mathbf{q}}_t, x)$, and thus prove the correctness of our local algorithm which sweeps all the vectors $\tilde{\mathbf{q}}_t$ instead of $\mathbf{q}_t$.

Finally, we use a simple trace lower bound to serve as the lower bound for $J(\mathbf{q}_t, x)$ and obtain the spectral characterization of the dense bipartite profile.

1.3 Related Works

Our work is closely related to a line of research on the *conductance* of a set S, which is defined as $\phi(S) = \frac{e(S,\bar{S})}{\min\{\mathrm{vol}(S), \mathrm{vol}(\bar{S})\}}$. Kannan, Vempala and Veta [13] suggest using the conductance as a measure of a set being a general community (in contrast of cyber-communities), since the smaller the conductance it, the more likely that the set is a community with dense intra-connections and sparse inter-connections. Spielman and Teng give the first local clustering algorithm to find subgraphs with small conductance by using the truncated random walk [29,30]. Anderson, Chung and Lang [5], Anderson and Peres [6], Kwok and Lau [16] and

Oveis Gharan and Trevisan [25] then give local algorithms for conductance with better approximation ratio or running time. All their local algorithms are based on the Cheeger's inequality that relates the second smallest eigenvalue of $\mathcal{L}$ to the conductance [2,1,28], similar to our algorithms which depend on Trevisan's spectral inequality that relates the largest eigenvalue of $\mathcal{L}$ to the B-ratio.

Some works studied the *small set expander*, that is, to find small set with small conductance. This problem is of interest not only for the reason that it has applications in finding small communities in social networks, but also that it is closely related to the unique games conjecture [27]. Arora, Barak and Steurer [7], Louis, Raghavendra, Tetali and Vempala [21], Lee, Oveis Gharan and Trevisan[17], Kwok and Lau [16], Oveis Gharan and Trevisan [25] and O'Donnell and Witmer [24] have given spectra based approximation algorithms and characterizations of this problem. The latter three works have recently shown that for any $0 < \epsilon < 1$,

$$\phi(\mathrm{vol}(G)/k^{1-\epsilon}) \le O(\sqrt{\lambda_k \log_k n}),$$

where $\phi(k)$ is the *expansion profile* of G and is defined as $\phi(k) := \min_{S:\mathrm{vol}(S)\le k} \phi(S)$. Their spectral characterization of the expansion profile as well as the Cheeger's inequality all use the first k smallest eigenvalues of $\mathcal{L}$, which is comparable to our characterization of the dense bipartite profile by the kth largest eigenvalue of $\mathcal{L}$ as given in inequality (2).

Feige, Kortsarz and Peleg [12], and Bhaskara et al. [10] give non-local algorithms for the densest k-subgraphs. Charikar [11], Andersen [3], Andersen and Chellapilla [4], Khuller and Saha[14] studied approximation (or local) algorithms for dense subgraphs based on other measures. Arora et al. [8] and Balcan et al. [9] investigate the problem of finding overlapping communities in networks.

All the proofs in the paper can be found in the arxiv version.

2 Preliminaries

Let $G = (V, E)$ be an undirected weighted graph and let $n := |V|$ and $m := |E|$. Let $d(v)$ denote the weighted degree of vertex v. For any vertex subset $S \subseteq V$, let $\bar{S} := V \backslash S$ denote the complementary of S. Let $e(S)$ be the number of edges in S and define the volume of S to be the sum of degree of vertices in S, that is $\mathrm{vol}(S) := \sum_{v\in S} d(v)$. Let $\mathrm{vol}(G) := \mathrm{vol}(V) = 2m$. For any two subsets $L, R \subseteq V$, let $e(L, R)$ denote the number of edges between L and R. For two disjoint subsets L, R, that is, $L \cap R = \emptyset$, we will use $U = (L, R)$ to denote subgraph induced on L and R, which is also called the pair subgraph. We will also use U to denote $L \cup R$. Given $U = (L, R)$, the *bipartiteness ratio (or B-ratio) of* U is defined as

$$\beta(L, R) := \frac{2e(L) + 2e(R) + e(U, \bar{U})}{\mathrm{vol}(U)}.$$

The *B-ratio of a set* S is defined to be the minimum value of $\beta(L, R)$ over all the possible partitions L, R of S, that is, $\beta(S) := \min_{(L,R)\text{ partition of } S} \beta(L, R)$.

The B-ratio of the graph G is defined as $\beta(G) := \min_{S\subseteq V} \beta(S)$.

We are interested in finding small subgraphs with small B-ratio. In the following, we use lower bold letters to denote vectors. Unless otherwise specified, a vector $\mathbf{p}$ is considered to be a row vector, and $\mathbf{p}^T$ is its transpose. For a vector $\mathbf{p}$ on vertices, let $\text{supp}(\mathbf{p})$ denote the support of $\mathbf{p}$, that is, the set of vertices on which the $\mathbf{p}$ value is nonzero. Let $\|\mathbf{p}\|_1$ and $\|\mathbf{p}\|_2$ denote the L^1 and L^2 norm of $\mathbf{p}$, respectively. Let $|\mathbf{p}|$ denotes its absolute vector, that is, $|\mathbf{p}|(v) := |\mathbf{p}(v)|$. For a vector $\mathbf{p}$ and a vertex subset S, let $\mathbf{p}(S) := \sum_{v \in S} \mathbf{p}(v)$. For L, R, let $\mathbf{p}(L, -R) := \sum_{v \in L} p(v) - \sum_{v \in R} p(v)$. One useful observation is that for any partition (L, R) of S, $\mathbf{p}(L, -R) \leq |\mathbf{p}|(S)$. Also note that there exists a partition (L_0, R_0) of S such that $\mathbf{p}(L_0, -R_0) = |\mathbf{p}|(S)$. Actually, L_0 is the set of vertices with positive $\mathbf{p}$ value and R_0 is the set of the remaining vertices, that is, $L_0 = \{v \in S : \mathbf{p}(v) > 0\}$ and $R_0 = \{v \in S : \mathbf{p}(v) \leq 0\}$.

For any vertex v, let χ_v denote the indicator vector on v. Now let A denote the adjacency matrix of the graph such that A_{uv} is the weight of edge $u \sim v$. Let D denote the diagonal degree matrix. Define the *random walk matrix* W, the *(normalized) Laplacian matrix* $\mathcal{L}$ and the *quasi-Laplacian matrix* M of the graph G as $W := D^{-1}A, \mathcal{L} := I - D^{-1/2}AD^{-1/2}, M := I - D^{-1}A$. It is well known that these three matrices are closely related. In particular, for regular graphs, $\mathcal{L}$ and M are the same; and if we let $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$ be the eigenvalues of $\mathcal{L}$, then $\{1 - \lambda_i\}_{0 \leq i \leq n-1}$ and $\{\lambda_i\}_{0 \leq i \leq n-1}$ are the eigenvalues of W and M, respectively. Note that for general graphs, though the eigenvalues of $\mathcal{L}$ and M are the same, the corresponding eigenvectors might be different. In this paper, we will mainly use the quasi-Laplacian M to give both algorithms and spectral characterization for the small dense bipartite subgraph problem. If we let $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{n-1}$ be the corresponding left eigenvectors of M, then we have the following spectral inequality given by Trevisan [33] (see also [26] as Trevisan did not explicitly state his result in terms of the matrix M).

Lemma 1 ([33]). *Let $\beta(G)$, λ_{n-1} and $\boldsymbol{v}_{n-1}$ defined as above. We have that $\beta(G) \leq \sqrt{2(2 - \lambda_{n-1})}$. Furthermore, a pair subgraph (X, Y) with B-ratio at most $\sqrt{2(2 - \lambda_{n-1})}$ can be found by a sweep process over $\boldsymbol{v}_{n-1}$.*

The sweep process mentioned above is defined as follows.

Definition 1. *(Sweep process) Given a vector $\boldsymbol{p}$, the sweep (process) over $\boldsymbol{p}$ is defined by performing the following operations:*

1. *Order the vertices so that $\frac{|p(v_1)|}{d(v_1)} \geq \frac{|p(v_2)|}{d(v_2)} \geq \cdots \geq \frac{|p(v_n)|}{d(v_n)}$.*
2. *For each $i \leq n$, let $L_i(\boldsymbol{p}) := \{v_j : \boldsymbol{p}(v_j) > 0 \, and j \leq i\}$, $R_i(\boldsymbol{p}) := \{v_j : \boldsymbol{p}(v_j) \leq 0 \, and j \leq i\}$ and $S_i(\boldsymbol{p}) := (L_i(\boldsymbol{p}), R_i(\boldsymbol{p}))$, which we call the sweep set of the first i vertices. Compute the B-ratio of $S_i(\boldsymbol{p})$.*

In Trevisan's inequality, to find the subgraph with small B-ratio, we just need to output the sweep set with the minimum B-ratio over the all the sweep sets. Trevisan also showed the tightness (within constant factors) of this inequality in the sense that there exist graphs such that the two quantities in both hands of the inequality in Lemma 1 are asymptotically the same. The sweep process as well as Trevisan's inequality are the bases of our algorithms for the small dense bipartite-like subgraphs.

We will use the following truncation operator to design local algorithms.

Definition 2. *(Truncation operator) Given a vector* $\boldsymbol{p}$ *and a nonnegative real number* ξ*, we define the* ξ*-truncated vector of* $\boldsymbol{p}$ *to be:*

$$[\boldsymbol{p}]_\xi(u) = \begin{cases} \boldsymbol{p}(u) & \text{if } |\boldsymbol{p}(u)| \geq \xi d(u), \\ 0 & \text{otherwise.} \end{cases}$$

3 Approximation Algorithm for the Small Dense Bipartite-Like Subgraphs

In this section, we first give the description of our approximation algorithm for the small dense bipartite-like subgraph, the main subroutine of which is the sweep process over a set of vectors $\chi_v M^t$. We then introduce a potential function $J(\mathbf{p}, x)$ and give both upper bound and lower bound of the potential function $J(\chi_v M^t, x)$ under certain conditions, using which we are able to show the correctness of our algorithm and thus prove Theorem 1.

Now we describe our algorithm `SwpDB` (short for "sweep for dense bipartite") for finding the small dense bipartite-like subgraphs.

```
SwpDB($G, k, \theta, \epsilon$)
Input: A graph $G$, a target volume $k > 4$, a target B-ratio $\theta$, an error parameter
$0 < \epsilon < 1/2$.
Output: A subgraph $(X, Y)$.

1. Let $T = \frac{\epsilon \ln 2k}{2\theta}$. Let $K = 2k^{1+\epsilon}$.
2. Sweep over all vectors $\chi_v M^t$, for each vertex $v \in V$ and $t \leq T$, to obtain a
   family $\mathcal{F}$ of sweep sets with volume at most $K$.
3. Output the subgraph $(X, Y)$ with the smallest B-ratio ratio among all sets
   in $\mathcal{F}$.
```

3.1 A Potential Function

We define a potential function $J : [0, 2m] \to \mathbb{R}^+$:

$$J(\mathbf{p}, x) := \max_{\substack{\mathbf{w} \in [0,1]^n \\ \sum_{v \in V} \mathbf{w}(v) d(v) = x}} \sum_{v \in V} |\mathbf{p}(v)| \mathbf{w}(v).$$

Note that our potential function is similar to a potential function for bounding the convergence of $\mathbf{p}(\frac{I+W}{2})^t$ in terms of the conductance given by Lovász and Simonovits [22,23]. Here we will use $J(\mathbf{p}, x)$ to bound the convergence of $\mathbf{q}M^t$ in terms of the B-ratio of the sweep sets.

There are two useful ways to see this potential function.

1. We view each edge $u \sim v \in E$ as two directed edges $u \to v$ and $v \to u$. For each directed edge $e = u \to v$, let $\mathbf{p}(e) = \frac{\mathbf{p}(u)}{d(u)}$. Order the edges so that$|\mathbf{p}(e_1)| \geq |\mathbf{p}(e_2)| \geq \cdots \geq |\mathbf{p}(e_{2m})|$.
   Now we can see that for an integer x, $J(\mathbf{p}, x) = \sum_{j=1}^{x} |\mathbf{p}(e_j)|$. For other fractional $x = \lfloor x \rfloor + r$, $J(\mathbf{p}, x) = (1-r)J(\mathbf{p}, \lfloor x \rfloor) + rJ(\mathbf{p}, \lceil x \rceil)$.
   Also it is easy to see that for any directed edge set F, $|\mathbf{p}|(F) := \sum_{e \in F} |\mathbf{p}|(e) \leq J(\mathbf{p}, |F|)$, since the former is a sum of $|\mathbf{p}|$ values of one specific set of edges with $|F|$ edges and the latter is the maximum over all such possible edge sets.
2. Another way to view the potential function is to use the sweep process over $\mathbf{p}$ as in Definition 1. By the definitions of the potential function and the sweep process, we have the following observations.
   (a) For $x = \mathrm{vol}(S_i(\mathbf{p}))$, $J(\mathbf{p}, x) = \sum_{j=1}^{i} |\mathbf{p}(v_j)| = |\mathbf{p}|(S_i(\mathbf{p})) = \mathbf{p}(L_i(\mathbf{p}), -R_i(\mathbf{p}))$. And $J(\mathbf{p}, x)$ is linear in other values of x.
   (b) For any set S, $|\mathbf{p}|(S) \leq J(\mathbf{p}, \mathrm{vol}(S))$, since the former is the sum of $|\mathbf{p}(v)|/d(v)$ values of vertices in S and the latter is the maximum sum over all sets with $|S|$ vertices;

From both views, we can easily see that the potential function is a non-decreasing and concave function of x.

3.2 An Upper Bound for the Potential Function

Now we upper bound $J(\mathbf{p}M, x)$ in terms of $J(\mathbf{p}, x')$ and the B-ratio of the sweep set of $\mathbf{p}M$.

Lemma 2 (Convergence Lemma). *For an arbitrary vector $\boldsymbol{p}$ on vertices, if $\beta(L_i(\boldsymbol{p}), R_i(\boldsymbol{p})) \geq \Theta$, then for $x = vol(S_i(\boldsymbol{p}))$, we have $J(\boldsymbol{p}M, x) \leq J(\boldsymbol{p}, x + \Theta x) + J(\boldsymbol{p}, x - \Theta x)$.*

We remark that the proof heavily depends on the definition and combinatorial property of the B-ratio of a set. Though the form is similar to the corresponding characterization of conductance given by Lovász and Simonovits, the two proofs are very different.

Now we can use the convergence lemma to upper bound $J(\chi_v M^t, x)$.

Lemma 3. *For any vertex $v \in V$, let $\boldsymbol{q}_t = \chi_v M^t$, if for all $t \leq T$ and all sweep sets $S_i(\boldsymbol{q}_t) = (L_i(\boldsymbol{q}_t), R_i(\boldsymbol{q}_t))$ of volume at most K have B-ratio at least Θ, that is, $\beta(L_i(\boldsymbol{q}_t), R_i(\boldsymbol{q}_t)) \geq \Theta$, then for any $t \leq T$, we have $J(q_t, x) \leq \frac{2^t x}{K} + \sqrt{\frac{x}{d(v)}}\left(2 - \frac{\Theta^2}{4}\right)^t$.*

3.3 A Lower Bound for the Potential Function

We show that if the graph contains a pair subgraph with small B-ratio, then we can have a good lower bound on $J(\chi_v M^t)$ for some vertex v. The following lemma is similar to the upper bounds on the escaping probability of random walks given by Oveis Gharan and Trevisan [25]. Our proof uses a new spectral analysis by using the *H-norm* of a vector and may be modified to give a different proof for the corresponding results in [25].

Lemma 4. *If $U = (L, R)$ has B-ratio $\beta(L, R) \leq \theta$, then for any integer $t > 0$,*

1. *there exists a vertex $v \in U$ such that $|\boldsymbol{q}_t|(U) \geq (2 - 2\theta)^t$, where $\boldsymbol{q}_t = \chi_v M^t$;*
2. *there exists a subset $U^t \subseteq U$ with $vol(U^t) \geq vol(U)/2$ satisfying that for any $v \in U^t$ and $\boldsymbol{q}_t = \chi_v M^t$, $J(\boldsymbol{q}_t, vol(U)) \geq |\boldsymbol{q}_t|(U) \geq \frac{1}{400}(2 - 6\theta)^t$, where we have assumed that $\theta < 1/3$.*

Now Theorem 1 can be proved by combining the Lemma 3 and 4.

By using a simple trace bound, we can obtain the following corollary that gives a spectral characterization of the small dense bipartite-like subgraphs and thus establish inequality (2).

Corollary 1. *If $\lambda_{n-k} \geq 2 - 2\eta$, then there is a polynomial time algorithm such that for any $0 < \epsilon < 1$, it finds a subset (X, Y) of volume at most $O(vol(G)/k^{1-\epsilon})$ and B-ratio $O(\sqrt{16(\eta/\epsilon)\log_k n})$.*

4 A Local Algorithm for Dense Bipartite-Like Subgraphs

We will use the truncated operation to give our local algorithm LocDB (short for "local algorithm for dense bipartite subgraph").

```
LocDB($G, v, k, \theta, \epsilon$)
Input: A graph $G$, a vertex $v$, a target volume $k > 2560000$, a target B-ratio
$\theta < 1/3$ and an error parameter $0 < \epsilon < 1/2$.
Output: A subgraph $(X, Y)$.

 1. Let $T = \frac{\epsilon \ln 1600k}{6\theta}$. Let $\xi_0 = \frac{k^{-1-\epsilon}}{800T}$, $\xi_t = \xi_0 2^t$. Let $\tilde{\mathbf{q}}_0 := \chi_v$, $\mathbf{r}_0 := [\tilde{\mathbf{q}}_0]_{\xi_0}$. Let
    $\mathcal{F} = \emptyset$.
 2. For each time $1 \leq t \leq T$:
    (a) Compute $\tilde{\mathbf{q}}_t := \mathbf{r}_{t-1}M$, $\mathbf{r}_t := [\tilde{\mathbf{q}}_t]_{\xi_t}$;
    (b) Sweep over the support of $\tilde{\mathbf{q}}_t$ and add to $\mathcal{F}$ all the sweep sets.
 3. Output the subgraph $(X, Y)$ with the smallest B-ratio ratio among all sets
    in $\mathcal{F}$.
```

Note that in the algorithm we just sweep the *support* of a given vector, which is important for the computation to be local.

Inspired by the proof of the correctness of SwpDB, we will use the upper bound and lower bound of the potential function $J(\tilde{\mathbf{q}}_t, x)$ to show the correctness of the local algorithm. Such bounds can be obtained by combining the following properties of the truncation operations in the algorithm.

Proposition 1. *For any vertex v, if $\boldsymbol{q}_t = \chi_v M^t$ and $\tilde{\boldsymbol{q}}_t, \boldsymbol{r}_t$ are as defined in the algorithm* LocDB*, then for any $t \geq 0$,*

1. *$\|\tilde{\boldsymbol{q}}_t\|_1 \leq 2^t$;*
2. *$|\boldsymbol{r}_t - \boldsymbol{q}_t| \leq \xi_0 t 2^t \boldsymbol{d}$, where $\boldsymbol{d}$ is the degree vector.*

Note that the second part of Proposition 1 directly implies a lower bound on $J(\tilde{\mathbf{q}}_t, x)$. More specifically, we have the following corollary.

Corollary 2. *For any set U, $|\tilde{\boldsymbol{q}}_t|(U) \geq |\boldsymbol{r}_t|(U) \geq |\boldsymbol{q}_t|(U) - \xi_0 t 2^t vol(U)$.*

We can also give an upper bound on $J(\tilde{\mathbf{q}}_t, x)$.

Lemma 5. *For any vertex v, $T > 0$, $\Theta < 1$, if for any $t \leq T$, the sweep sets $S_i(\tilde{\boldsymbol{q}}_t)$ of volume at most K have B-ratio at least Θ, then for any $0 \leq t \leq T$ and $0 \leq x \leq 2m$,*

$$J(\tilde{\boldsymbol{q}}_t, x) \leq \frac{2^t x}{K} + \sqrt{\frac{x}{d(v)}}\Big(2 - \frac{\Theta^2}{4}\Big)^t.$$

Now by using Corollary 2 and Lemma 5, we can show the correctness of the algorithm `LocDB` and thus prove Theorem 2.

References

1. Alon, N.: Eigenvalues and expanders. Combinatorica 6(2), 83–96 (1986)
2. Alon, N., Milman, V.: λ_1, isoperimetric inequalities for graphs, and superconcentrators. Journal of Combinatorial Theory, Series B 38(1), 73–88 (1985)
3. Andersen, R.: A local algorithm for finding dense subgraphs. ACM Transactions on Algorithms (TALG) 6(4), 60 (2010)
4. Andersen, R., Chellapilla, K.: Finding dense subgraphs with size bounds. In: Avrachenkov, K., Donato, D., Litvak, N. (eds.) WAW 2009. LNCS, vol. 5427, pp. 25–37. Springer, Heidelberg (2009)
5. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vectors. In: IEEE Symposium on Foundations of Computer Science (2006)
6. Andersen, R., Peres, Y.: Finding sparse cuts locally using evolving sets. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (2009)
7. Arora, S., Barak, B., Steurer, D.: Subexponential algorithms for unique games and related problems. In: Foundations of Computer Science, FOCS (2010)
8. Arora, S., Ge, R., Sachdeva, S., Schoenebeck, G.: Finding overlapping communities in social networks: toward a rigorous approach. In: Proceedings of the 13th ACM Conference on Electronic Commerce (2012)
9. Balcan, M.F., Borgs, C., Braverman, M., Chayes, J., Teng, S.: Finding endogenously formed communities. In: SODA (2013)
10. Bhaskara, A., Charikar, M., Chlamtac, E., Feige, U., Vijayaraghavan, A.: Detecting high log-densities: an $o(n^{1/4})$ approximation for densest k-subgraph. In: Proceedings of the 42nd ACM Symposium on Theory of Computing (2010)
11. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 84–95. Springer, Heidelberg (2000)
12. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica 29(3), 410–421 (2001)
13. Kannan, R., Vempala, S., Vetta, A.: On clusterings: Good, bad and spectral. J. ACM 51(3), 497–515 (2004)
14. Khuller, S., Saha, B.: On finding dense subgraphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 597–608. Springer, Heidelberg (2009)

15. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the web for emerging cyber-communities. In: Proceedings of the 8th International Conference on World Wide Web, pp. 1481–1493 (1999)
16. Kwok, T.C., Lau, L.C.: Finding small sparse cuts by random walk. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 615–626. Springer, Heidelberg (2012)
17. Lee, J., Oveis Gharan, S., Trevisan, L.: Multi-way spectral partitioning and higher-order cheeger inequalities. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (2012)
18. Leskovec, J., Lang, K., Dasgupta, A., Mahoney, M.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6(1), 29–123 (2009)
19. Leskovec, J., Lang, K.J., Mahoney, M.: Empirical comparison of algorithms for network community detection. In: Proceedings of the 19th International Conference on World Wide Web, pp. 631–640 (2010)
20. Liao, C.S., Lu, K., Baym, M., Singh, R., Berger, B.: IsoRankN: spectral methods for global alignment of multiple protein networks. Bioinformatics, i253–i258 (2009)
21. Louis, A., Raghavendra, P., Tetali, P., Vempala, S.: Many sparse cuts via higher eigenvalues. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (2012)
22. Lovász, L., Simonovits, M.: The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In: Foundations of Computer Science (FOCS), pp. 346–354 (1990)
23. Lovász, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm. Random structures & Algorithms 4(4), 359–412 (1993)
24. O'Donnell, R., Witmer, D.: Improved small-set expansion from higher eigenvalues. Arxiv preprint arXiv:1204.4688 (2012)
25. Oveis Gharan, S., Trevisan, L.: Approximating the expansion profile and almost optimal local graph clustering. In: 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS (2012)
26. Peng, P.: A local algorithm for finding dense bipartite-like subgraphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 145–156. Springer, Heidelberg (2012)
27. Raghavendra, P., Steurer, D.: Graph expansion and the unique games conjecture. In: 42nd ACM Symposium on Theory of Computing, pp. 755–764 (2010)
28. Sinclair, A., Jerrum, M.: Approximate counting, uniform generation and rapidly mixing markov chains. Information and Computation 82(1), 93–133 (1989)
29. Spielman, D., Teng, S.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, pp. 81–90 (2004)
30. Spielman, D., Teng, S.: A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. arXiv:0809.3232 (2008)
31. Spielman, D.A.: Algorithms, graph theory, and linear equations. In: Proceedings of the International Congress of Mathematicians IV, pp. 2698–2722 (2010)
32. Teng, S.-H.: The laplacian paradigm: Emerging algorithms for massive graphs. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 2–14. Springer, Heidelberg (2010)
33. Trevisan, L.: Max cut and the smallest eigenvalue. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 263–272 (2009)

Approximate Čech Complex in Low and High Dimensions

Michael Kerber[1] and R. Sharathkumar[2]

[1] Stanford University, Stanford, USA and Max Planck Center for Visual Computing and Communication, Saarbrücken, Germany
mkerber@mpi-inf.mpg.de
[2] Stanford University, Stanford, USA
sharathk@stanford.edu

Abstract. Čech complexes reveal valuable topological information about point sets at a certain scale in arbitrary dimensions, but the sheer size of these complexes limits their practical impact. While recent work introduced approximation techniques for filtrations of (Vietoris-)Rips complexes, a coarser version of Čech complexes, we propose the approximation of Čech filtrations directly.

For fixed dimensional point set S, we present an approximation of the Čech filtration of S by a sequence of complexes of size linear in the number of points. We generalize well-separated pair decompositions (WSPD) to well-separated simplicial decomposition (WSSD) in which every simplex defined on S is covered by some element of WSSD. We give an efficient algorithm to compute a linear-sized WSSD in fixed dimensional spaces. Using a WSSD, we then present a linear-sized approximation of the filtration of Čech complex of S.

We also present a generalization of the known fact that the Rips complex approximates the Čech complex by a factor of $\sqrt{2}$. We define a class of complexes that interpolate between Čech and Rips complexes and that, given any parameter $\varepsilon > 0$, approximate the Čech complex by a factor $(1+\varepsilon)$. Our complex can be represented by $O(n^{\lceil 1/2\varepsilon \rceil})$ simplices, up to purely combinatorial operations, without any hidden dependence on the ambient dimension of the point set. Our results are based on an interesting link between Čech complex and coresets for minimum enclosing ball of high-dimensional point sets. As a consequence of our analysis, we show improved bounds on coresets that approximate the radius of the minimum enclosing ball.

1 Introduction

Motivation. A common theme in topological data analysis is the analysis of point cloud data representing an unknown manifold. Although the ambient space can be high-dimensional, the manifold itself is usually of relatively low dimension. Manifold learning techniques try to infer properties of the manifold, like its dimension or its homological properties, from the point sample.

An early step in this pipeline is to construct a cell complex from the point sample which shares similarities with the hidden manifold. The *Čech complex at scale* α (with $\alpha \geq 0$) captures the intersection structure of balls of radius α centered at the input points. More precisely, it is the *nerve* of these balls, and is therefore homotopically equivalent to their union. Increasing α from 0 to ∞ yields a *filtration*, a sequence of nested Čech complexes, which can serve as the basis of multi-scale approaches for topological data analysis.

A notorious problem with Čech complexes is their representation: Its k-skeleton can consist of up to $O(n^k)$ simplices, where n is the number of input points. Moreover, its construction requires the computation of minimum enclosing balls of point sets; we will make this relation explicit in Section 2. A common workaround is to replace the Čech complex by the *(Vietoris-)Rips complex* at the same scale α. Its definition only depends on the diameter of point sets and can therefore be computed by only looking at the pairwise distances (therefore, unlike the Čech complex, it is defined for arbitrary metrics). Although Rips complexes permit a sparser representation, they do not resolve the issue that the final complex can consist of a large number of simplices; Sheehy [20] and Dey et al. [8] have recently addressed this problem by defining an approximate Rips filtration whose size is only linear in the input size. On the other hand, efficient methods for approximating minimum enclosing balls have been established, even for high-dimensional problems, whereas the diameter of point sets appears to be a significantly harder problem in an approximate context. This suggests that Čech complexes might be more suitable objects than Rips complexes in an approximate context, at least when the point set is embedded in Euclidean space.

Contribution. We give two different approaches to approximate filtrations of Čech complexes, both connecting the problem to well-known concepts in discrete geometry: The first approach yields, for a fixed constant dimension, a sequence of complexes, each of linear size in the number of input points, that approximate the Čech filtration. By approximate, we mean that the persistence modules induced by exact and approximate Čech filtration are $(1+\varepsilon)$-interleaved (see Section 2). To achieve this result, we generalize the famous *well-separated pair decomposition (WSPD)* to a higher-dimensional analogue, that we call the *well-separated simplicial decomposition (WSSD).* Intuitively, a WSSD decomposes a point set S into a linear number of tuples with respect to the number of points. A k-tuple in the WSSD can be viewed as k clusters of points of S with the property that whenever a ball contains at least one point of each cluster, a small expansion of the ball contains all points in all clusters. Furthermore, these tuples cover every simplex with vertices in S, i.e., given any k-simplex σ, there is a $k+1$-tuple of clusters such that each cluster contains one vertex of σ. We consider the introduction of WSSDs to be of independent interest: given the numerous applications of WSPD, we hope that its generalization will find further applications in approximate computational topology. We finally remark that the constant in the size of our filtration depends exponentially on the dimension of the ambient space, which restricts the applicability to low-dimensional spaces;

this is a major difference to related work on the Rips filtration [20,8] which depends only on the *doubling dimension* of the point set.

As our second contribution, we prove a generalized version of the well-known Vietoris-Rips lemma [10, p.62] which states that the Čech complex at scale α is contained in the Rips complex at scale $\sqrt{2}\alpha$. We define a family of complexes, called *completion complexes* such that for any ε, the Čech complex at scale α is contained in a completion complex at scale $(1+\varepsilon)\alpha$. These completions complexes are parametrized by an integer k; the k-completion is completely determined by its k-skeleton, consisting of up to $O(n^k)$ complexes, in the sense that higher-dimensional simplices can be obtained by combinatorial operations only. To achieve $(1+\varepsilon)$-closeness to the Čech complex, we need to set $k \approx 1/(2\varepsilon)$ (see Theorem 4 for the precise statement); in particular, there is no dependence on the ambient dimension to approximate the Čech complex arbitrarily closely.

For proving this result, we use *coresets* for minimum enclosing ball (meb) [3]: the meb of a set of points can be approximated by selecting only a small subset of the input which is called a *coreset*; here approximation means that an ε-expansion of the meb of the coreset contains all input points. The size of the smallest coreset is at most $\lceil 1/\varepsilon \rceil$, independent of the number of points and the ambient dimension, and this bound is tight [3]. To obtain our result, we relax the definition of coreset for minimum enclosing balls. We only require the *radius* of the meb to be approximated, not the meb itself. We prove that even smaller coresets of size roughly $\lceil 1/(2\varepsilon) \rceil$ always exist for approximating the radius of the meb. Again, we consider this coreset result to be of independent interest.

Related Work. Sparse representation of complexes based on point cloud data are a popular subject in current research. Standard techniques are the *alpha complex* [11,12] which contains all Delaunay simplices up to a certain circumradius (and their faces), *simplex collapses* which remove a pair of simplices from the complex without changing the homotopy type (see [1,17,21] for modern references), and *witness approaches* which construct the complex only on a small subset of landmark points and use the other points as witnesses [7,2,9]. A more extensive treatment of some of these techniques can be found in [10, Ch.III]. Another very recent approach [19] constructs Rips complexes at several scales and connects them using *zigzag persistence* [5], an extension to standard persistence which allows insertions and deletions in the filtration. The aforementioned work by Sheehy [20] combines this theory with *net-trees* [14], a variant of hierarchical metric spanners, to get an approximate linear-size zigzag-filtration of the Rips complex in a first step and finally shows that the deletions in the zigzag can be ignored. Dey et al. [8] arrive at the same result more directly by constructing an hierarchical ε-net, defining a filtration from it where the elements are connected by simplicial maps instead of inclusions, and finally showing that this filtration is *interleaved* with the Rips-filtration in the sense of [6].

Outline. We introduce basic topological concepts in Section 2. Then we discuss WSSDs, our generalization of WSPDs and give an algorithm to compute them in Section 3. We show how to use WSSDs to approximates the persistence diagram of the Čech complex in Section 4. The existence of small coresets for approximating the radius of meb and the generalized Vietoris-Rips Lemma are presented in Section 5.

Due to space restrictions, we had to remove most of the proofs and some further explanations. We refer to [16] for the full version of the paper.

2 Preliminaries

Simplicial Complexes. Let S denote a finite set of universal elements, called *vertices*[1]. A *(simplicial) complex* C is a collection of subsets of S, called *simplices*, with the property that whenever a simplex σ is in C, all its (non-empty) subsets are in C as well. These non-empty subsets are called the *faces* of σ; a *proper face* is a face that is not equal to σ. Setting $k := \|\sigma\| - 1$, where $\|\cdot\|$ stands for the number of elements considered as a subset, we call σ a *k-simplex.* Observe a k-simplex σ corresponds to a $(k+1)$-subset $(v_0, \ldots, v_k)$ of S; these $(k+1)$ vertices are called the *boundary vertices* of k-simplex, and we will frequently identify the simplex and its boundary vertices. The *k-skeleton* of a complex C is the set of all ℓ-simplices in C with $\ell \leq k$. Let K and K' be two simplicial complexes with vertex sets V and V' and consider a map $f : V \to V'$. If for any simplex $(v_0, \ldots, v_k)$ of K, $(f(v_0), \ldots f(v_k))$ yields a simplex in K', then f extends to a map from K to K' which we will also denote by f; in this case, f is called a *simplicial map.*

For a finite point set P and $\alpha > 0$, the *Čech complex* $\mathcal{C}_\alpha(P)$ is the *nerve* of the set of (closed) balls of radius α centered at the points in P, which means that every ball is represented by a vertex, and a k-simplex is in $\mathcal{C}_\alpha(P)$ if the corresponding balls have a common intersection. Note that a k-simplex of the Čech complex can be identified with $(k+1)$ points $p_0, \ldots, p_k$ in P. Let $\mathrm{meb}(p_0, \ldots, p_k)$ denote the *minimum enclosing ball of P*, that is, the ball with minimal radius that contains each p_i. Then, a k-simplex $\{p_0, \ldots, p_k\}$ is in $\mathcal{C}_\alpha(P)$ if and only if the radius of $\mathrm{meb}(p_0, \ldots, p_k)$ is at most α.

A widely used approximation of Cech complexes is the *(Vietoris)-Rips complex* $\mathcal{R}_\alpha(P)$. We define it inductively in dimension: It has the same 1-skeleton as the Cech complex, and a k-simplex is in $\mathcal{R}_\alpha$ if all its faces are in $\mathcal{R}_\alpha$. The Rips complex is an example of a *clique complex* (also known as *flag complex* or *Whitney complex*). That means, it is completely determined by its 1-skeleton which in turn only depends on the pairwise distance between the input points. For $k+1$ points $p_0, \ldots, p_k$ in P, let the *diameter* $\mathrm{diam}(p_0, \ldots, p_k)$ denote the maximal pairwise distance between any two points p_i and p_j with $0 \leq i \leq j \leq k$. Then, a k-simplex $\{p_0, \ldots, p_k\}$ is $\mathcal{R}_\alpha(P)$ if and only if $\mathrm{diam}(p_0, \ldots, p_k)$ is at most α. For notational convenience, we will often omit the P from the notation and write $\mathcal{C}_\alpha$ and $\mathcal{R}_\alpha$ when P is clear from context.

[1] The finiteness of S is assumed for the sake of simplicity, but not necessary for most definitions.

Persistence Modules. For $A \subset \mathbb{R}$, a *persistent module* is a family $(F_\alpha)_{\alpha \in A}$ of vector spaces with homomorphisms $f_\alpha^{\alpha'} : F_\alpha \to F_{\alpha'}$ for any $\alpha \leq \alpha'$ such that $f_{\alpha'}^{\alpha''} \circ f_\alpha^{\alpha'} = f_\alpha^{\alpha''}$ and f_α^α is the identity function.[2]. The most common class are modules induced by a *filtration*, that is, a family of complexes $(C_\alpha)_{\alpha \in A}$ such that $C_\alpha \subseteq C_{\alpha'}$ for $\alpha \leq \alpha'$. For some fixed dimension p, set $H_\alpha := H_p(C_\alpha)$, where $H_p(K)$ denotes the *p-th homology group* of the complex K; see [10] or [18] for a full treatment of this concept. The inclusion map from C_α to $C_{\alpha'}$ induces an homomorphism $\hat{f}_\alpha^{\alpha'} : H_\alpha \to H_{\alpha'}$ and turns $(H_\alpha)_{\alpha \in \mathbb{R}}$ into a persistence module. Example of such filtrations and their induced modules are the *Cech filtration* $(\mathcal{C}_\alpha)_{\alpha \geq 0}$ and the Rips filtration $(\mathcal{R}_\alpha)_{\alpha \geq 0}$. However, we will also consider persistence modules which are not induced by filtrations. We will frequently denote filtrations and modules by F_* instead of $(F_\alpha)_{\alpha \in A}$ if there is no confusion about A.

An *ε-interleaving* between two persistence modules $(F_\alpha)_{\alpha \geq 0}$ and $(G_\alpha)_{\alpha \geq 0}$ is given by two families of homomorphisms $\{\phi : F_\alpha \to G_{c\alpha}\}_{\alpha \geq 0}$ and $\{\psi : G_\alpha \to F_{c\alpha}\}_{\alpha \geq 0}$ such that all the following diagrams commute for any $c \geq 1$ and $\alpha, \alpha' \geq 0$:

$$\begin{tikzcd}
F_{\frac{\alpha}{c}} \arrow[rrr] \arrow[dr] & & & F_{c\alpha'} \\
& G_\alpha \arrow[r] & G_{\alpha'} \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
& F_{c\alpha} \arrow[rr] & & F_{c\alpha'} \\
G_\alpha \arrow[ur] \arrow[rr] & & G_{\alpha'} \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
& F_\alpha \arrow[r] & F_{\alpha'} \arrow[dr] & \\
G_{\frac{\alpha}{c}} \arrow[ur] \arrow[rrr] & & & G_{c\alpha'}
\end{tikzcd}$$

$$\begin{tikzcd}
F_\alpha \arrow[rr] \arrow[dr] & & F_{\alpha'} \arrow[dr] & \\
& G_{c\alpha} \arrow[rr] & & G_{c\alpha'}
\end{tikzcd}$$

Interleavings define a similarity measure between modules in the following sense: Every persistence module can be fully described by its *persistence diagram* which tracks the creation and destruction of homological features when increasing the scale parameter of the module. An ε-interleaving between two modules implies that their persistence diagrams are approximations of each other; see [6][10] for more details. In the case of modules induced by filtrations $(A_\alpha)_{\alpha \geq 0}$ and $(B_\alpha)_{\alpha \geq 0}$, if $c > 1$ is such that $A_{\frac{\alpha}{c}} \subset B_\alpha \subset A_{c\alpha}$ for any $\alpha \geq 0$, the induced persistence modules are c-interleaved [20].

3 Well-Separated Simplicial Decompositions

In this section, we introduce the notion of Well-separated simplicial decomposition (WSSD) of point sets. WSSD can be seen as a generalization of well-separated pair decomposition of a point set. We first revisit the definition of WSPD and then generalize it to WSSD.

[2] This is not the most general definition of a persistent module; see [6]

Notations. Let $S \subset \mathbb{R}^d$ be a fixed point set with minimal distance $1/\sqrt{d}$ between two points and such that all points are contained in a axis-parallel hypercube q with side length 2^L. We consider a *quadtree* Q of q where each node represents a hypercube; the root represents q, and when an internal node represents a hypercube q', its children represent the hypercubes obtained by splitting q' into 2^d congruent hypercubes. From now on, we will usually identify a quadtree node and the hypercube that it represents. We call a node of Q *empty* if it does not contain any point of S. For any internal node q', the *height* of q' in Q is i if the side length of q' is 2^i; the construction ends at height 0; by construction, each leaf contains at most one point of S.[3] The nodes of Q at height i induce a grid G_i where the side length of every cell of G_i is 2^i. For $e > 0$ and a ball $\mathbb{B}$ with center c and radius r, we let $e\mathbb{B}$ denote the ball with center c and radius $e \cdot r$. Finally, we assume the parameter ε to be in $(0, 1)$ from now.

Well-Separated Pair Decomposition. Let Q be a quadtree for S. A pair of quadtree cells (q, q') is called *ε-well separated* if $\max(\mathrm{diam}(q), \mathrm{diam}(q')) \leq \varepsilon d(q, q')$; here $\mathrm{diam}(q)$ is the diameter of a quadtree cell (which equals $2^h\sqrt{d}$ if h is the height of q) and $d(q, q')$ is the closest distance between cells q and q'. For a pair $(p, p') \in S \times S$ we say that a pair of quadtree cells (q, q') *covers* (p, p') if $p \in q$ and $p' \in q'$, or $p \in q'$ and $p' \in q$. An *ε-well separated pair decomposition* (ε-WSPD) of S is a set of pairs $\Gamma = ((q_1, q_1'), (q_2, q_2'), \ldots, (q_m, q_m'))$ such that all pairs are ε-well separated and every edge in $S \times S$ is covered by some pair in Γ. A WSPD of size $O(n/\varepsilon^d)$ can be computed in $O(n \log n + n/\varepsilon^d)$ time as proved first in [4]; see also [13, §3] for a modern treatment:

Well-Separated Simplicial decomposition. We generalize the construction of WSPD to higher dimensions: Let S and Q be as above. We call a $(k + 1)$-tuple $(q_0, \ldots, q_k)$ of quadtree cells an *ε-well separated tuple* (ε-WST), if for any ball $\mathbb{B}$ that contains at least one point of each q_ℓ, we have that

$$q_0 \cup q_1 \cup \ldots q_k \subseteq (1 + \varepsilon)\mathbb{B}. \tag{3.1}$$

Moreover, we say that $(q_0, \ldots, q_k)$ *covers* a k-simplex $\sigma = (p_0, \ldots, p_k)$, $p_0, \ldots, p_k \in S$ if there is a permutation π of $(0, \ldots, k)$ such that $p_{\pi(\ell)} \in q_\ell$ for all $0 \leq \ell \leq k$.

Definition 1. *A set of $(k+1)$-tuples $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ is a (ε, k)-*well separated simplicial decomposition *((ε, k)-WSSD), if each γ_ℓ is a ε-well separated tuple and each k-simplex of S is covered by some γ_ℓ. An ε-*WSSD *is the union of (ε, k)-WSSDs over all $1 \leq k \leq d$.*

Our Algorithm. We present a recursive algorithm for computing an (ε, k)-WSSD. If $k = 1$, we use the algorithm from [13, Fig. 3.3] to compute an $\frac{\varepsilon}{2}$-WSPD, which is an $(\varepsilon, 1)$-WSSD. If $k > 1$, we recursively compute an $(\varepsilon, k-1)$-WSSD Γ_{k-1} and

[3] This "construction" is only conceptual; in an actual implementation, only non-empty would be stored. Moreover, the quadtree should be represented in *compressed* form to avoid dependence on the *spread* of the point set; see [13, §2] for details.

construct an (ε, k)-WSSD Γ_k as follows: We initialize Γ_k as the empty set and iterate over the elements in Γ_{k-1}. For an ε-WST $\gamma = (q_0, q_1, \ldots q_{k-1}) \in \Gamma_{k-1}$, let $\mathbb{B}_\gamma = \text{meb}(q_0 \cup q_1 \cup \ldots q_{k-1})$, and let r denote its radius. Consider the grid G_h formed by all quadtree cells of height h such that $2^h \leq \frac{\varepsilon r}{2\sqrt{d}} \leq 2^{h+1}$. We compute the set of non-empty quadtree cells in G_h that intersect the ball $2 \cdot \mathbb{B}_\gamma$. For each such cell q', we add the $(k+1)$-tuple $(q_0, \ldots, q_{k-1}, q')$ to Γ_k.

Correctness. In order to prove the correctness of our construction procedure, we need to show that the generated tuples indeed form a (ε, k)-WSSD.

First, we show that every tuple added by our procedure is an ε-WST. We do induction on k, noting that for $k = 1$, the statement is true because an $\frac{\varepsilon}{2}$-WSPD is an $(\varepsilon, 1)$-WSSD. For $k \geq 2$, assume that our algorithm creates a k-tuple $(q_0, \ldots, q_{k-1}, q')$ by adding the cell q' while considering the ε-WST $(q_0, \ldots, q_{k-1})$. Let $\mathbb{B}$ be a ball that contains at least one point from each of the cells $(q_0, \ldots, q_{k-1}, q')$. We have to argue that $(1+\varepsilon)\mathbb{B}$ contains the cells $q_0, \ldots, q_{k-1}, q'$; by induction hypothesis, it is clear that $q_0 \cup \ldots \cup q_{k-1} \subseteq (1+\varepsilon)\mathbb{B}$ and moreover, $r = \text{rad}(q_0, \ldots, q_{k-1}) \leq (1+\varepsilon)\text{rad}(\mathbb{B})$. By construction,

$$\text{diam}(q') \leq \frac{\sqrt{d}\varepsilon r}{2\sqrt{d}} \leq \frac{\varepsilon(1+\varepsilon)\text{rad}(\mathbb{B})}{2} \leq \varepsilon \cdot \text{rad}(\mathbb{B}).$$

It follows that $q' \subseteq (1+\varepsilon)\mathbb{B}$.

Second, we show that the set of $(k+1)$-tuples Γ_k generated by our procedure covers all k-simplices over S. Again, we do induction on k. For the base case $k = 1$, by definition, all pairs of points in $S \times S$ are covered by some pair (q, q') in an $\frac{\varepsilon}{2}$-WSPD. Assume that the computed $(\varepsilon, k-1)$-WSSD covers all $(k-1)$-simplices and consider any k-simplex $\sigma = (p_0, \ldots, p_k)$. There exists a point among the p_i, say p_0, such that $p_0 \in 2\text{meb}(\sigma')$ [3], where $\sigma' = (p_1, \ldots, p_k)$. By induction hypothesis, there exists a ε-WST $t = (q_1, \ldots, q_k)$ that covers σ'. Clearly, $p_0 \in 2\text{meb}(t)$ as well. Let q be the cell of G_h that contains p_0. By construction, our algorithm adds $(q_1, \ldots, q_k, q)$ to Γ_k, and this tuple covers σ.

Analysis. The size of the (ε, k)-WSSD generated by our algorithm is bounded by $n(d/\varepsilon)^{O(dk)}$, as one can show quite easily by induction on k, bounding the number of grid cells considered in the induction step by a simple packing argument. The running time is proportional to the number of cells visited. This yields a running time of $O(|\Gamma_{k-1}|(d/\varepsilon)^d) = O(n(d/\varepsilon)^{O(dk)})$ for computing Γ_k from Γ_{k-1}, and a total running time of $O(n \log n + n(d/\varepsilon)^{O(dk)})$ It follows that the total running time for computing $\Gamma_1, \ldots, \Gamma_k$ is bounded by $O(n \log n + n(d/\varepsilon)^{O(dk)})$, taking the WSPD construction time into account.

Doubling Dimension. Similar as in the WSPD case, we believe that the (exponential) dependance of WSSD on the ambient dimension can be improved to the doubling dimension of the finite metric space induced by the point set, by replacing the quadtree-based construction by an approach using net-trees [14]. However, several non-trivial adaptations are needed for this improvement, and we postpone a thorough discussion to an extended version of the paper.

4 Čech Approximations of Linear Size

In this section, we will define a persistence module which is a $(1+\varepsilon)$-approximation of the Čech module in the sense of Section 2. Due to space restrictions, we will only outline the construction, omitting any proofs and additional remarks; we again refer to the arxiv version [16] for more details.

For a finite point set $S \subset \mathbb{R}^d$, consider a quadtree Q and an $\frac{\varepsilon}{5}$-WSSD defined over cells of Q, computed with the algorithm from Section 3. Having a WST $t = (q_0, \ldots, q_k)$ of that WSSD, we write $\mathrm{rad}(t)$ for the radius of the minimum enclosing ball of $q_0 \cup \ldots \cup q_k$. For a non-empty quadtree cell q, we choose a *representative* $\mathrm{rep}(q)$ in S with the property that if q is internal, its representative is chosen among the representatives of its children. Moreover, for any quadtree cell q of height i or less, we define $\mathrm{qcell}(q, i)$ for its (unique) ancestor at level i.

We fix the following additional parameters: Set $\theta_\ell := (1+\varepsilon)^\ell$ for any integer ℓ. Let Δ_α denote the integer such that

$$\theta_{\Delta_\alpha} \leq \alpha < \theta_{\Delta_\alpha+1}.$$

Furthermore, we define h_α as the integer such that

$$2^{h_\alpha} \leq \frac{2\varepsilon\theta_{\Delta_\alpha}}{5\sqrt{d}} < 2^{h_\alpha+1}.$$

When there is no ambiguity about α, we will just write $\Delta := \Delta_\alpha$ and $h := h_\alpha$.

The Approximation Complex. Recall that G_ℓ denotes the set of all quadtree cells at height ℓ. We construct a simplicial complex $\mathcal{A}_\alpha$ over the vertex set G_h (with $h := h_\alpha$) in the following way: For any WST $t' = (q_0, \ldots, q_k)$ with all q_i at height h or less, let $t = (\mathrm{qcell}(q_0, h), \ldots, \mathrm{qcell}(q_k, h))$. If $\mathrm{rad}(t) \leq (1 + \frac{2}{3}\varepsilon)\theta_\Delta$, we add the simplex t to $\mathcal{A}_\alpha$. Note that some of the $\mathrm{qcell}(q_\ell, h)$ can be the same, so that the resulting simplex might be of dimension less than k. We can show that $\mathcal{A}_\alpha$ is indeed a simplicial complex. It is clear by construction and by the analysis of the previous section that $\mathcal{A}_\alpha$ consists of at most $n(d/\varepsilon)^{O(d^2)}$ simplices,

We define maps between the $\mathcal{A}_\alpha$ next: Consider two scales $\alpha_1 < \alpha_2$. We set $h_1 := h_{\alpha_1}$ and define h_2, Δ_1, and Δ_2 accordingly. Since $h_1 \leq h_2$, there is a natural map $g_{\alpha_1}^{\alpha_2} : G_{h_1} \to G_{h_2}$, mapping a quadtree cell at height h_1 to its ancestor at height h_2. This naturally extends to a map $g_{\alpha_1}^{\alpha_2} : \mathcal{A}_{\alpha_1} \to \mathcal{A}_{\alpha_2}$, by mapping a simplex $\sigma = (v_0, \ldots, v_k)$ to $g_{\alpha_1}^{\alpha_2}(\sigma) := (g_{\alpha_1}^{\alpha_2}(v_0), \ldots, g_{\alpha_1}^{\alpha_2}(v_k))$. We can show that $g_{\alpha_1}^{\alpha_2}$ is a simplicial map, that $g_{\alpha'}^{\alpha''} \circ g_{\alpha}^{\alpha'} = g_{\alpha}^{\alpha''}$ and $g_\alpha^\alpha = \mathrm{id}$.

Next, we investigate the *cross-map* $\phi : \mathcal{C}_{\frac{\alpha}{1+\varepsilon}} \to \mathcal{A}_\alpha$. To define it for a vertex $v \in \mathcal{C}_{\frac{\alpha}{1+\varepsilon}}$ (which is a point of S), set $\phi(v) = q$, where q is the quadtree cell at level h that contains v. For a simplex $(v_0, \ldots, v_k)$, define $\phi(v_0, \ldots, v_k) = (\phi(v_0), \ldots, \phi(v_k))$. In the other direction, we have a map $\psi : \mathcal{A}_\alpha \to \mathcal{C}_{(1+\varepsilon)\alpha}$ defined by mapping a quadtree cell q at level h to its representative $\mathrm{rep}(q)$. We can show that ψ and ϕ are simplicial.

We fix some integer $p \geq 0$ and consider the persistence modules

$$(\hat{\mathcal{C}}_\alpha)_{\alpha \geq 0} := (H_p(\mathcal{C}_\alpha))_{\alpha \geq 0}, \quad (\hat{\mathcal{A}}_\alpha)_{\alpha \geq 0} := (H_p(\mathcal{A}_\alpha))_{\alpha \geq 0},$$

where $H_p(\cdot)$ is the p-th homology group over an arbitrary base field, with the induced homomorphisms $\hat{f}_{\alpha_1}^{\alpha_2}$ (induced by inclusion) and $\hat{g}_{\alpha_1}^{\alpha_2}$, respectively. The cross-maps $\hat{\phi}$, $\hat{\psi}$ commute with the module maps $\hat{f}$, $\hat{g}$ in the sense that the four diagrams from Section 2 all commute. This shows our approximation result:

Theorem 1. *The persistence module $\hat{\mathcal{A}}_*$ and $\hat{\mathcal{C}}_*$ are $(1+\varepsilon)$-interleaved.*

5 Coresets for Minimal Enclosing Ball Radii and a Generalized Vietoris-Rips Lemma

Recall that for a point set $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$, we denote by meb$(P)$ the *minimum enclosing ball of* P. Let center$(P) \in \mathbb{R}^d$ denote the center and rad$(P) \geq 0$ the radius of meb(P). Fixing $\varepsilon > 0$, we call a subset $C \subseteq P$ a *meb-coreset for* P if the ball centered at center(C) and with radius $(1+\varepsilon)$rad(C) contains P. We call $C \subseteq P$ a *radius-coreset for* P if rad$(P) \leq (1+\varepsilon)$rad(C). Informally, a radius-coreset approximates only the radius of the minimum enclosing ball, whereas the meb-coreset approximates the ball itself.

It is straight-forward to verify that a meb-coreset is also a radius-coreset by definition, but the opposite is not always the case: for instance, in an equilateral triangle in the plane, one subset of 2 points forms a radius coreset for $\varepsilon = 0.5$, but the subset cannot be a meb-coreset for the same ε-value.

Obviously, a point set is a coreset of itself, so coresets exist for any point set. We are interested in the coresets of small sizes. For the meb-coreset, this question is answered by Bădoiu and Clarkson [3]: For $\varepsilon > 0$, and any (finite) point set, there exists a meb-coreset of size $\lceil \frac{1}{\varepsilon} \rceil$, and there exist point sets where any meb-coreset has size at least $\lceil \frac{1}{\varepsilon} \rceil$. Note that the size of the coreset is independent of both the number of points in P and the ambient dimension. However, since radius-coresets are a relaxed version of meb-coresets, we can hope for even smaller coresets. We define

$$\delta := \lceil \frac{1}{2\varepsilon + \varepsilon^2} + 1 \rceil$$

Theorem 2. *There is a point set such that any radius-coreset has size at least δ.*

Proof. Consider the standard simplex in d dimensions (with d to be fixed later), that is, P is the point set given by the d unit vectors in $\mathbb{R}^d$. By elementary calculations, it can be verified that center$(P) = (\frac{1}{d}, \dots, \frac{1}{d})$ and rad$(P) = \sqrt{\frac{d-1}{d}}$. Fixing a subset $C \subseteq P$ of size k, its points span a standard simplex in $\mathbb{R}^k$ and therefore, rad$(C) = \sqrt{\frac{k-1}{k}}$ by the same argument. Hence, C is a radius-coreset of P if and only if $\sqrt{\frac{d-1}{d}} \leq (1+\varepsilon)\sqrt{\frac{k-1}{k}}$. Isolating k yields the equivalent condition that

$$k \geq \lceil \frac{(1+\varepsilon)^2}{(1+\varepsilon)^2 - \frac{d-1}{d}} \rceil = \lceil 1 + \frac{1}{\frac{d}{d-1}(2\varepsilon + \varepsilon^2 + \frac{1}{d})} \rceil.$$

The last expression is monotonously increasing in d, and converges to δ for $d \to \infty$. It follows that, for d large enough, any radius-coreset has size at least δ.

We will show next that any point set has a radius coreset of size δ. For a point set P in $\mathbb{R}^d$ and $1 \leq k \leq d$, let $r_k(P)$ denote the maximal radius of a meb among all subsets of P of cardinality k. We can assume that P contains at least $d+1$ points; otherwise it is contained in a lower-dimensional Euclidean space. On the other hand, if P contains at least $d+1$ points, there exists a subset P' of P containing exactly $d+1$ points such that the meb of P' equals the meb of P, which implies that $r_{d+1}(P) = \mathrm{rad}(P)$.

Theorem 3. *For $\varepsilon > 0$, any finite point set P has a radius-coreset of size δ.*

Proof. We use a theorem by Henk [15, Thm.1] which proves that for $2 \leq j \leq i \leq d+1$, it holds that $r_i(P) \leq \sqrt{\frac{j(i-1)}{i(j-1)}} r_j(P)$. Applying this result to the case that $i = d+1$ and $j = \delta$ yields

$$\mathrm{rad}(P) = r_{d+1}(P) \leq \sqrt{\frac{\delta \cdot d}{(d+1)(\delta-1)}} r_\delta(P) = \underbrace{\sqrt{\frac{d}{d+1}}}_{\leq 1} \sqrt{\frac{\delta}{\delta-1}} r_\delta(P).$$

Furthermore, since $\delta \geq \frac{1}{2\varepsilon+\varepsilon^2} + 1$, it follows that $\frac{\delta}{\delta-1} = 1 + \frac{1}{\delta-1} \leq (1+\varepsilon)^2$. So, letting C be a subset of cardinality δ with radius $r_\delta(P)$, we obtain that $\mathrm{rad}(P) \leq (1+\varepsilon)\mathrm{rad}(C)$, which means that C is a radius-coreset.

We next define the following generalization of a flag-complex: The *i-completion* $\mathcal{M}_i(K)$ of a simplicial complex K, is the maximal complex whose i-skeleton equals the i-skeleton of K. With that notation, we have that $\mathcal{R}_\alpha = \mathcal{M}_1(\mathcal{C}_\alpha)$. Moreover, we have that $\mathcal{C}_\alpha = \mathcal{M}_d(\mathcal{C}_\alpha)$ as an immediate consequence of Helly's Theorem [10]. We can show the following result as a consequence of Theorem 3.

Theorem 4. *For $\delta = \lceil 1/(2\varepsilon + \varepsilon^2) + 1 \rceil$, $\mathcal{C}_\alpha \subseteq \mathcal{M}_{\delta-1}(\mathcal{C}_\alpha) \subseteq \mathcal{C}_{(1+\varepsilon)\alpha}$.*

Proof. The first inclusion is clear. Now, consider a simplex σ in $\mathcal{M}_{\delta-1}(\mathcal{C}_\alpha)$. The second inclusion is trivial if $\dim \sigma \leq \delta - 1$, so let its dimension be at least δ. By Theorem 3, the boundary vertices of σ have a coreset of size at most δ. Let τ denote the simplex spanned by such a coreset. As τ is a face of σ, it is contained in $\mathcal{M}_{\delta-1}(\mathcal{C}_\alpha)$, and because it is of dimension $\delta - 1$, it is in particular contained in $C(\alpha)$. By the property of coresets, the minimal enclosing ball of σ has radius at most $(1+\varepsilon)\alpha$ which implies that $\sigma \in \mathcal{C}_{(1+\varepsilon)\alpha}$.

For $\varepsilon = \sqrt{2} - 1$, we get $\delta = 2$, and Theorem 4 yields the Vietoris-Rips Lemma as stated in [10, p.62]. Moreover, Theorem 4 asserts that the persistence module of $\mathcal{M}_{\delta-1}(\mathcal{C}_*)$ is $(1+\varepsilon)$-interleaved with the persistence diagram of $\mathcal{C}_*$.

Acknowledgements. We thank Okke Schrijvers for helpful discussions. The first author acknowledges support by the Max Planck Center for Visual Computing and Communication. The second author acknowledges financial support from grant #FA9550-12-1-XXXX from the U.S. Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Projects Agency (DARPA).

References

1. Attali, D., Lieutier, A., Salinas, D.: Efficient data structures for representing and simplifying simplicial complexes in high dimensions. Int. J. of Computational Geometry & Applications 22, 279–303 (2012)
2. Boissonnat, J.-D., Guibas, L., Oudot, S.: Manifold reconstruction in arbitrary dimensions using witness complexes. Discr. Comput. Geom. 42, 37–70 (2009)
3. Bădoiu, M., Clarkson, K.: Optimal core-sets for balls. Computational Geometry: Theory and Applications 40, 14–22 (2008)
4. Callahan, P., Kosaraju, S.: A decomposition of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields. J. of the ACM 42, 67–90 (1995)
5. Carlsson, G., de Silva, V.: Zigzag persistence. Foundations of Computational Mathematics 10, 367–405 (2010)
6. Chazal, F., Cohen-Steiner, D., Glisse, M., Guibas, L., Oudot, S.: Proximity of persistence modules and their diagrams. In: Proc. 25th ACM Symp. on Comp. Geom., pp. 237–246 (2009)
7. de Silva, V., Carlsson, G.: Topological estimation using witness complexes. In: Symp. on Point-Based Graphics, pp. 157–166 (2004)
8. Dey, T., Fan, F., Wang, Y.: Computing topological persistence for simplicial maps. CoRR, abs/1208.5018 (2012)
9. Dey, T., Fan, F., Wang, Y.: Graph induced complex on point data. In: Proc. 29th ACM Symp. on Comp. Geom. (2013)
10. Edelsbrunner, H., Harer, J.: Computational Topology, An Introduction. American Mathematical Society (2010)
11. Edelsbrunner, H., Kirkpatrick, D., Seidel, R.: On the shape of a set of points in the plane. IEEE Trans. Inform. Theory IT-29, 551–559 (1983)
12. Edelsbrunner, H., Mücke, E.: Three-dimensional alpha shapes. ACM Trans. Graphics 13, 43–72 (1994)
13. Har-Peled, S.: Geometric Approximation Algorithms. American Mathematical Society (2011)
14. Har-Peled, S., Mendel, M.: Fast construction of nets in low dimensional metrics and their applications. Siam J. on Computing 35, 1148–1184 (2006)
15. Henk, M.: A generalization of Jung's theorem. Geometriae Dedicata 42, 235–240 (1992)
16. Kerber, M., Sharathkumar, R.: Approximate cech complexes in low and high dimensions. arxiv:1307.3272 (2013)
17. Mrozek, M., Pilarczyk, P., Zelazna, N.: Homology algorithm based on acyclic subspace. Computer and Mathematics with Applications 55, 2395–2412 (2008)
18. Munkres, J.R.: Elements of algebraic topology. Westview Press (1984)
19. Oudot, S., Sheehy, D.: Zigzag zoology: Rips zigzags for homology inference. In: Proc. 29th ACM Symp. on Comp. Geom. (2013)
20. Sheehy, D.: Linear-size approximation to the Vietoris-Rips filtration. In: Proc. 2012 ACM Symp. on Comp. Geom., pp. 239–248 (2012)
21. Zomorodian, A.: The tidy set: A minimal simplicial set for computing homology of clique complexes. In: Proc. 26th ACM Symp. on Comp. Geom., pp. 257–266 (2010)

Model Counting for Formulas of Bounded Clique-Width[⋆]

Friedrich Slivovsky and Stefan Szeider

Institute of Information Systems, Vienna University of Technology, Vienna, Austria
fs@kr.tuwien.ac.at, stefan@szeider.net

Abstract. We show that #SAT is polynomial-time tractable for classes of CNF formulas whose incidence graphs have bounded symmetric clique-width (or bounded clique-width, or bounded rank-width). This result strictly generalizes polynomial-time tractability results for classes of formulas with signed incidence graphs of bounded clique-width and classes of formulas with incidence graphs of bounded modular treewidth, which were the most general results of this kind known so far.

1 Introduction

Propositional model counting (#SAT) is the problem of computing the number of satisfying truth assignments for a given CNF formula. It is a well-studied problem with applications in Artificial Intelligence, such as probabilistic inference [1, 16]. It is also a notoriously hard problem: #SAT is #P-complete in general [18] and remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas [14]. It is NP-hard to approximate the number of satisfying truth assignments of a formula with n variables to within $2^{n^{1-\varepsilon}}$ for any $\varepsilon > 0$. As in the exact case, this hardness result even holds for monotone 2CNF formulas and Horn 2CNF formulas [14]. While these syntactic restrictions do not make the problem easier, #SAT becomes tractable under certain *structural* restrictions [6, 8, 9, 11–13, 15, 17]. Structural restriction are obtained by bounding parameters of (hyper)graphs associated with formulas. We extend this line of research and study #SAT for classes of formulas whose incidence graphs (that is, the bipartite graph whose vertex classes consist of variables and clauses, with variables adjacent to clauses they occur in) have bounded *symmetric clique-width* [4]. Symmetric clique-width is a parameter that is closely related to clique-width, rank-width, and Boolean-width: a class of graphs has bounded symmetric clique-width *iff* it has bounded clique-width *iff* it has bounded rank-width *iff* it has bounded Boolean-width. For a graph class $\mathcal{C}$, let #SAT($\mathcal{C}$) be the restriction of #SAT to instances F with incidence graph $I(F) \in \mathcal{C}$. We prove:

Theorem 1. *#SAT($\mathcal{C}$) is polynomial-time tractable for any graph class $\mathcal{C}$ of bounded symmetric clique-width.*

[⋆] This research was supported by the ERC (COMPLEX REASON, 239962).

$$\begin{tikzcd}
 & \textbf{symmetric incidence clique-width} \text{ [this paper]} & \\
\text{signed incidence clique-width [6]} \arrow[ur] & & \text{modular incidence treewidth [13]} \arrow[ul] \\
 & \text{incidence treewidth [6, 15, 17]} \arrow[ul] \arrow[ur] & \\
 & \text{primal treewidth [15]} \arrow[u] &
\end{tikzcd}$$

Fig. 1. A hierarchy of structural parameters. An arc from a parameter p to a parameter q reads as "for any class of formulas, q is bounded whenever p is bounded." Bold type is used to indicate parameters that render #SAT polynomial-time tractable when bounded by a constant.

This result generalizes polynomial-time tractability results for classes of formulas with signed incidence graphs of bounded clique-width [6] and classes of formulas with incidence graphs of bounded modular treewidth [13]. The situation is illustrated in Figure 1 (for a survey of results for width-based parameters, see [12, 13]). Our result is obtained through a combination of dynamic programming on a decomposition tree with the representation of truth assignments by *projections* (i.e., sets of clauses satisfied by these assignments). This extends the techniques used to prove polynomial-tractability of #SAT for classes of formulas with incidence graphs of bounded modular treewidth [13]; there, partial assignments are partitioned into equivalence classes by an equivalence relation roughly defined as follows: two assignments are equivalent whenever they satisfy the same set of clauses of a certain formula induced by a subtree of the decomposition. To make bottom-up dynamic programming work, it is enough to record the number of assignments in each equivalence class. This approach does not carry over to the case of bounded symmetric clique-width for principal reasons: the number of equivalence classes of such a relation can be exponential in the size of the (sub)formula.

To deal with this, our algorithm uses the technique of taking into account an "expectation from the outside" [2, 7, 8]. The underlying idea is that the information one has to record for any particular partial solution can be reduced significantly if one includes an "expectation" about what this partial solution will be combined with to form a complete solution. This trick allows us to bound the number of records required for dynamic programming by a polynomial in the number of clauses of the input formula.

For all parameters considered in Figure 1, propositional model counting is polynomial-time tractable if the parameter is bounded by a constant, but some of them even admit so-called FPT algorithms. The runtime of an FPT algorithm is bounded by a function of the form $f(k)p(l)$, where f is an arbitrary computable function and p is a polynomial with order independent of the parameter k. As we will see, the order of the polynomial bounding the runtime in Theorem 1 is dependent on the parameter. One may wonder whether this can be avoided, that is, whether the problem admits an FPT algorithm. The following result shows that this is not possible, subject to an assumption from parameterized complexity.

Theorem 2 ([12]). SAT*, parameterized by the symmetric clique-width of the incidence graph of the input formula, is* W[1]*-hard.*

To be precise, the result proven in [12] is stated in terms of clique-width. However, since the symmetric clique-width of a graph with clique-width k is at most 2^k (see [4]), the result carries over to symmetric clique-width.

2 Preliminaries

Let $f : X \to Y$ be a function and $X' \subseteq X$. We let $f(X') = \{\, f(x) \in Y : x \in X' \,\}$. Let X^* and Y^* be sets, and let $g : X^* \to Y^*$ be a function with $g(x) = f(x)$ for all $x \in X \cap X^*$. Then the function $f \cup g : X \cup X^* \to Y \cup Y^*$ is defined as $(f \cup g)(x) = f(x)$ if $x \in X$ and $(f \cup g)(x) = g(x)$ if $x \in X^* \setminus X$.

Graphs. The graphs considered in this paper are loopless, simple, and undirected. If G is a graph and v is a vertex of G, we let $N(v)$ denote the set of all neighbors of v in G. For a tree T we write $L(T)$ to denote the set of leaves of T. Let $\mathcal{C}$ be a class of graphs and let f be a mapping (invariant under isomorphisms) that associates each graph G with a non-negative real number. We say $\mathcal{C}$ *has bounded* f if there is a c such that $f(G) \leq c$ for every $G \in \mathcal{C}$.

Formulas. We assume an infinite supply of propositional *variables.* A *literal* is a variable x or a negated variable $\overline{x}$; we put $var(x) = var(\overline{x}) = x$; if $y = \overline{x}$ is a literal, then we write $\overline{y} = x$. For a set S of literals we write $\overline{S} = \{\, \overline{x} : x \in S \,\}$; S is *tautological* if $S \cap \overline{S} \neq \emptyset$. A *clause* is a finite non-tautological set of literals. A finite set of clauses is a *CNF formula* (or *formula*, for short). The *length* of a formula F is given by $\sum_{C \in F} |C|$. A variable x *occurs* in a clause C if $x \in C \cup \overline{C}$. We let $var(C)$ denote the set of variables that occur in C. A variable x *occurs* in a formula F if it occurs in at least one of its clauses, and we let $var(F) = \bigcup_{C \in F} var(C)$. If F is a formula and X a set of variables, we let $F|_X = \{\, C \in F : X \subseteq var(C) \,\}$. The *incidence graph* of a formula F is the bipartite graph $I(F)$ with vertex set $var(F) \cup F$ and edge set $\{\, Cx : C \in F$ and $x \in var(C) \,\}$.

Let F be a formula. A *truth assignment* is a mapping $\tau : X \to \{0, 1\}$ defined on some set of variables $X \subseteq var(F)$. We call τ *total* if $X = var(F)$ and *partial* otherwise. For $x \in X$, we define $\tau(\overline{x}) = 1 - \tau(x)$. A truth assignment τ *satisfies* a clause C if C contains some literal ℓ with $\tau(\ell) = 1$. If τ satisfies all clauses of F, then τ *satisfies* F; in that case we call F satisfiable. The *Satisfiability* (SAT) problem is that of testing whether a given formula is satisfiable. The *propositional model counting* (#SAT) problem is a generalization of SAT that asks for the number of satisfying total truth assignments of a given formula. For a graph class $\mathcal{C}$, we let #SAT($\mathcal{C}$) be the restriction of #SAT to instances F with $I(F) \in \mathcal{C}$.

Decomposition Trees. We review decomposition trees following the presentation in [3]. Let $G = (V, E)$ be a graph. A *decomposition tree* for G is a pair (T, δ), where T is a rooted binary tree and $\delta : L(T) \to V$ is a bijection. For a subset $X \subseteq V$ let $\overline{X} = V \setminus X$. We associate every edge $e \in E(T)$ with a bipartition P_e of V obtained as follows. If T_1 and T_2 are the components obtained by removing e from T, we let $P_e = (L(T_1), L(T_2))$. Note that $L(T_2) = \overline{X}$ for $X = L(T_1)$. A function $f : 2^V \to \mathbb{R}$ is *symmetric* if $f(X) = f(\overline{X})$ for all $X \subseteq V$. Let $f : 2^V \to \mathbb{R}$ be a symmetric function. The f-width of (T, δ) is the maximum of $f(X) = f(\overline{X})$ taken over the bipartitions $P_e = (X, \overline{X})$ for all $e \in E(T)$. The f-width of G is the minimum of the f-widths of the decomposition trees of G.

Let $A(G)$ stand for the *adjacency matrix* of G, that is, the $V \times V$ matrix $A(G) = (a_{vw})_{v \in V, w \in V}$ such that $a_{vw} = 1$ if $vw \in E$ and $a_{vw} = 0$ otherwise. For $X, Y \subseteq V$, let $A(G)[X, Y]$ denote the $X \times Y$ submatrix $(a_{vw})_{v \in X, w \in Y}$. The *cut-rank* function $\rho_G : 2^V \to \mathbb{R}$ of G is defined as

$$\rho_G(X) = \mathit{rank}(A(G)[X, V \setminus X]),$$

where *rank* is the rank function of matrices over $\mathbb{Z}_2$. The row and column ranks of any matrix are equivalent, so this function is symmetric. The *rank-width* of a decomposition tree (T, δ) of G, denoted $\mathit{rankw}(T, \delta)$, is the ρ_G-width of (T, δ), and the *rank-width* of G, denoted $\mathit{rankw}(G)$, is the ρ_G-width of G.

Let X be a proper nonempty subset of V. We define an equivalence relation $\equiv_X$ on X as

$$x \equiv_X y \text{ iff, for every } z \in V \setminus X,\ xz \in E \Leftrightarrow yz \in E.$$

The *index* of X in G is the cardinality of $X/{\equiv_X}$, that is, the number of equivalence classes of $\equiv_X$. We let $\mathit{index}_G : 2^V \to \mathbb{R}$ be the function that maps each proper nonempty subset X of V to its index in G. We now define the function $\iota_G : 2^V \to \mathbb{R}$ as

$$\iota_G(X) = \max(\mathit{index}_G(X), \mathit{index}_G(V \setminus X)).$$

This function is trivially symmetric. The *index* of a decomposition tree (T, δ) of G, denoted $\mathit{index}(T, \delta)$, is the ι_G-width of (T, δ). The *symmetric clique-width* [4] of G, denoted $\mathit{scw}(G)$, is the ι_G-width of G.

Symmetric clique-width and rank-width are closely related graph parameters. In fact, the index of a decomposition tree can be bounded in terms of its rank-width.

Lemma 3. *For every graph G and decomposition tree (T, δ) of G, $\mathit{rankw}(T, \delta) \le \mathit{index}(T, \delta) \le 2^{\mathit{rankw}(T, \delta)}$.*

Corollary 4. *For every graph G, $\mathit{rankw}(G) \le \mathit{scw}(G) \le 2^{\mathit{rankw}(G)}$.*

Runtime bounds for the dynamic programming algorithm presented below are more naturally stated in terms the index of the underlying decomposition tree than in terms of its rank-width. However, to the best of our knowledge, there is no polynomial-time algorithm for computing decomposition trees of minimum index directly – instead, we will use the following result to compute decomposition trees of minimum rank-width.

Theorem 5 ([5]). *Let $k \in \mathbb{N}$ be a constant and $n \geq 2$. For an n-vertex graph G, we can output a decomposition tree of rank-width at most k or confirm that the rank-width of G is larger than k in time $O(n^3)$.*

Projections. Let F be a set of clauses and X a set of variables. For an assignment $\sigma \in 2^X$ we write $F(\sigma)$ to denote the set of clauses of F satisfied by σ, and call $F(\sigma)$ a *projection* of F. We write $\mathsf{proj}(F, X) = \{ F(\sigma) : \sigma \in 2^X \}$ for the set of projections of F with respect to a set X of variables.

Proposition 6. *Let F be a formula with m clauses and let $X \subseteq var(F)$ be a set of variables. We have $|\mathsf{proj}(F|_X, X)| \leq m + 1$. Moreover, the set $\mathsf{proj}(F|_X, X)$ can be computed in time polynomial in l, where l is the length of F.*

Proof. Let $\sim_X$ be the relation on clauses defined as $C \sim_X C'$ if $\{ \ell \in C : var(\ell) \in X \} = \{ \ell \in C' : var(\ell) \in X \}$. Clearly $\sim_X$ is an equivalence relation. Let $\mathcal{C}_1, \dots, \mathcal{C}_l$ be the equivalence classes of $\sim_X$ on $F|_X$. Recall that every clause C in $F|_X$ contains all variables in X. As a consequence, an assignment $\tau \in 2^X$ either satisfies all clauses in $F|_X$ or it satisfies all clauses in $F|_X$ except those in a unique class $\mathcal{C}_i$ for $i \in \{1, \dots, l\}$, in which case $F|_X(\tau) = F|_X \backslash \mathcal{C}_i$. Since $F|_X \subseteq F$ we get $l \leq m$, and thus $|\mathsf{proj}(F|_X, X)| \leq m + 1$. Computing $\mathsf{proj}(F|_X, X)$ boils down to computing $\mathcal{C}_1, \dots, \mathcal{C}_l$ and in turn $F|_X \setminus \mathcal{C}_i$ for each $i \in \{1, \dots, l\}$, which can be done in time polynomial in the length of F. The set $F|_X$ is contained in $\mathsf{proj}(F|_X, X)$ if and only if $l < 2^{|X|}$, which can be checked in polynomial time as well. □

3 An Algorithm for #SAT

In this section, we will describe an algorithm for #SAT via dynamic programming on a decomposition tree. Due to space constraints, several proofs are placed in the appendix. To simplify the statements of intermediate results, we fix a formula F with $|F| = m$ clauses and a decomposition tree (T, δ) of $I(F)$ with $index(T, \delta) = k$. For a node $z \in V(T)$, let T_z denote the maximal subtree of T rooted at z. We write var_z for the set of variables $var(F) \cap \delta(L(T_z))$ and F_z for the set of clauses $F \cap \delta(L(T_z))$. Moreover, we let $\overline{F_z} = F \setminus F_z$ and $\overline{var_z} = var(F) \setminus var_z$.

Our algorithm combines techniques from [13] with dynamic programming using "expectations" [2, 7, 8]. We briefly describe the information maintained for each node $z \in V(T)$ of the decomposition. Classes of truth assignments $\sigma \in 2^{var_z}$ will be represented by two sets of clauses. The first set (typically denoted *out*) corresponds to the projection $\overline{F_z}(\sigma)$, that is, the set of clauses *outside* the current subtree that is satisfied by σ. The second set is a projection $F_z(\tau)$ for some $\tau \in 2^{\overline{var_z}}$ so that the combined assignment $\sigma \cup \tau$ satisfies F_z. This set of clauses (typically denoted *in*) is "expected" to be satisfied from outside the current subtree by an "incoming" assignment. Adopting the terminology of [8], we call these pairs of sets *shapes*.

Definition 7 (Shape). *Let $z \in V(T)$, let $out_z \subseteq \overline{F_z}$, and let $in_z \subseteq F_z$. We call the pair (out_z, in_z) a* shape (for z)*, and say an assignment $\tau \in 2^{var_z}$ is* of shape (out_z, in_z) *if it satisfies the following conditions.*

(i) $\overline{F_z}(\tau) = out_z$.

(ii) For each clause $C \in F_z$, the assignment τ satisfies C or $C \in in_z$.

If $out_z \in \mathsf{proj}(\overline{F_z}, var_z)$ and $in_z \in \mathsf{proj}(F_z, \overline{var_z})$ then the shape (out_z, in_z) is proper*. We denote the set of shapes for $z \in V(T)$ by $\mathsf{shapes}(z)$ and write $N_z(s)$ to denote the set of assignments in 2^{var_z} of shape $s \in \mathsf{shapes}(z)$. Moreover, we let $n_z(s) = |N_z(s)|$.*

Note that an assignment can have multiple shapes, so shapes do not partition assignments into equivalence classes.

Lemma 8. *A truth assignment $\tau \in 2^{var(F)}$ satisfies F if and only if it has shape $(\emptyset, \emptyset)$. Moreover, the shape $(\emptyset, \emptyset)$ is proper.*

This tells us that $n_r((\emptyset, \emptyset))$ is equal to the number of satisfying truth assignments of F. Let $x, y, z \in V(T)$ such that x and y are the children of z, and let s_x, s_y, s_z be shapes for x, y, z, respectively. The assignments in $N_x(s_x)$ and $N_y(s_y)$ contribute to $N_z(s_z)$ if certain conditions are met. These are captured by the following definition.

Definition 9. *Let $x, y, z \in V(T)$ such that x and y are the children of z. We say two shapes $(out_x, in_x) \in \mathsf{shapes}(x)$ and $(out_y, in_y) \in \mathsf{shapes}(y)$* generate *the shape $(out_z, in_z) \in \mathsf{shapes}(z)$ whenever the following conditions are satisfied.*

(1) $out_z = (out_x \cup out_y) \cap \overline{F_z}$

(2) $in_x = (in_z \cup out_y) \cap F_x$

(3) $in_y = (in_z \cup out_x) \cap F_y$

We write $\mathsf{generators}_z(s)$ for the set of pairs in $\mathsf{shapes}(x) \times \mathsf{shapes}(y)$ that generate $s \in \mathsf{shapes}(z)$.

Lemma 10. *Let $x, y, z \in V(T)$ such that x and y are the children of z, and let $\tau_x \in 2^{var_x}$ be of shape $(out_x, in_x) \in \mathsf{shapes}(x)$ and $\tau_y \in 2^{var_y}$ be of shape $(out_y, in_y) \in \mathsf{shapes}(y)$. If (out_x, in_x) and (out_y, in_y) generate the shape $(out_z, in_z) \in \mathsf{shapes}(z)$, then $\tau = \tau_x \cup \tau_y$ is of shape (out_z, in_z). Moreover, if (out_z, in_z) is proper then (out_x, in_x) and (out_y, in_y) are proper.*

Proof. Suppose (out_x, in_x) and (out_y, in_y) generate (out_z, in_z). To see that τ satisfies condition (i), note that a clause is satisfied by τ if and only if it is satisfied by τ_x or τ_y, so $\overline{F_z}(\tau_z) = \overline{F_z}(\tau_x) \cup \overline{F_z}(\tau_y) = (out_x \cap \overline{F_z}) \cup (out_y \cap \overline{F_z}) = out_z$. For condition (ii), let $C \in F_z = F_x \cup F_y$. Without loss of generality assume that $C \in F_x$. Suppose τ does not satisfy C. Then τ_x does not satisfy C, so we must have $C \in in_x$ because τ_x is of shape (out_x, in_x). But τ_y does not satisfy C either, so $C \notin out_y$. Combining these statements, we get $C \in in_x \setminus out_y$. Because (out_x, in_x) and (out_y, in_y) generate (out_z, in_z) we have $in_x = (in_z \cup out_y) \cap F_x$ by condition (2). It follows that $C \in in_z$.

The assignments τ_x and τ_y are of shapes (out_x, in_x) and (out_y, in_y) so $out_x \in \mathsf{proj}(\overline{F_x}, var_x)$ and $out_y \in \mathsf{proj}(\overline{F_y}, var_y)$ by condition (i). Suppose (out_z, in_z) is proper. Then there is an assignment $\rho \in 2^{\overline{var_z}}$ such that $in_z = F_z(\rho)$. The shapes (out_x, in_x) and (out_y, in_y) generate (out_z, in_z), so $in_x = (in_z \cup out_y) \cap F_x$. Thus $in_x = (F_z(\rho) \cup \overline{F_y}(\tau_y)) \cap F_x$. Equivalently, $in_x = (F_z(\rho) \cap F_x) \cup (\overline{F_y}(\tau_y) \cap F_x)$. Since $F_x \subseteq F_z$ and $F_x \subseteq \overline{F_y}$ this can be rewritten once more as $in_x = F_x(\rho) \cup F_x(\tau_y)$. The domains $\overline{var_z}$ of ρ and var_y of τ_y are disjoint, so $F_x(\rho) \cup F_x(\tau_y) = F_x(\rho \cup \tau_y)$. Because $\overline{var_z} \cup var_y = \overline{var_x}$ it follows that $in_x \in \mathsf{proj}(F_x, \overline{var_x})$ and so (out_x, in_x) is proper. A symmetric argument shows that (out_y, in_y) is proper. □

Corollary 11. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$ be proper. Suppose $s_x \in \mathsf{shapes}(x)$ and $s_y \in \mathsf{shapes}(y)$ generate s and both $N_x(s_x)$ and $N_y(s_y)$ are nonempty. Then s_x and s_y are proper.*

Lemma 12. *Let $x, y, z \in V(T)$ such that x and y are the children of z, and let $\tau \in 2^{var_z}$ be a truth assignment of shape $(out_z, in_z) \in \mathsf{shapes}(z)$. Let $\tau_x = \tau|_{var_x}$ and $\tau_y = \tau|_{var_y}$. There are unique shapes $(out_x, in_x) \in \mathsf{shapes}(x)$ and $(out_y, in_y) \in \mathsf{shapes}(y)$ generating (out_z, in_z) such that τ_x has shape (out_x, in_x) and τ_y has shape (out_y, in_y).*

Proof. We define $out_x = \overline{F_x}(\tau_x)$, $out_y = \overline{F_y}(\tau_y)$ and let $in_x = (in_z \cap F_x) \cup F_x(\tau_y)$, $in_y = (in_z \cap F_y) \cup F_y(\tau_x)$. We prove that (out_x, in_x) and (out_y, in_y) generate (out_z, in_z). Since τ has shape (out_z, in_z) by condition (i) we have $out_z = \overline{F_z}(\tau)$. We further have $\overline{F_z}(\tau) = \overline{F_z}(\tau_x) \cup \overline{F_z}(\tau_y)$ by choice of τ_x and τ_y. Because $\overline{F_z} \subseteq \overline{F_x}$ and $\overline{F_z} \subseteq \overline{F_y}$ we get $\overline{F_z}(\tau) = (\overline{F_x}(\tau_x) \cap \overline{F_z}) \cup (\overline{F_y}(\tau_y) \cap \overline{F_z})$ and thus $\overline{F_z}(\tau) = (out_x \cup out_y) \cap \overline{F_z}$. That is, condition (1) is satisfied. From $F_x \subseteq \overline{F_y}$ and $F_y \subseteq \overline{F_x}$ it follows that $F_x(\tau_y) = \overline{F_y}(\tau_y) \cap F_x$ and $F_y(\tau_x) = \overline{F_x}(\tau_x) \cap F_y$. Thus $F_x(\tau_y) = out_y \cap F_x$ and $F_y(\tau_x) = out_x \cap F_y$ by construction of out_x and out_y. By inserting in the definitions of in_x and in_y we get $in_x = (in_z \cap F_x) \cup (out_y \cap F_x)$ and $in_y = (in_z \cap F_y) \cup (out_x \cap F_y)$, so conditions (2) and (3) are satisfied. We conclude that (out_x, in_x) and (out_y, in_y) generate (out_z, in_z).

We proceed to showing that τ_x is of shape (out_x, in_x). Condition (i) is satisfied by construction. To see that condition (ii) holds, pick any $C \in F_x$ not satisfied by τ_x. If τ_y satisfies C, then $C \in F_x(\tau_y) \subseteq in_x$. Otherwise, $\tau = \tau_x \cup \tau_y$ does not satisfy C. Since τ of shape (out_z, in_z) this implies $C \in in_z$. Again we get $C \in in_x$ as $in_z \cap F_x \subseteq in_x$. The proof that τ_y has shape (out_y, in_y) is symmetric.

To show uniqueness, let $(out'_x, in'_x) \in \mathsf{shapes}(x)$ and $(out'_y, in'_y) \in \mathsf{shapes}(y)$ generate (out_z, in_z), and suppose τ_x has shape (out'_x, in'_x) and τ_y has shape (out'_y, in'_y). From condition (i) we immediately get $out'_x = \overline{F_x}(\tau_x) = out_x$ and $out'_y = \overline{F_y}(\tau_y) = out_y$. Since the pairs (out'_x, in'_x),(out'_y, in'_y) and (out_x, in_x), (out_y, in_y) both generate (out_z, in_z), it follows from condition (2) that $in'_x = in_x$ and $in'_y = in_y$. □

Lemma 13. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$. The following equality holds.*

$$n_z(s) = \sum_{(s_x,s_y)\in \mathsf{generators}_z(s)} n_x(s_x)\, n_y(s_y) \tag{1}$$

Corollary 14. *Let $x, y, z \in V(T)$ such that x and y are the children of z in T, and let $s \in \mathsf{shapes}(z)$ be proper. Let $P = \{\, (s_x, s_y) \in \mathsf{generators}_z(s) : s_x$ and s_y are proper$\}$. The following equality holds.*

$$n_z(s) = \sum_{(s_x,s_y)\in P} n_x(s_x)\, n_y(s_y) \tag{2}$$

Proof. By Corollary 11 the product $n_x(s_x)n_y(s_y)$ is nonzero only if s_x and s_y are proper, for any pair $(s_x, s_y) \in \mathsf{generators}_z(s)$. In combination with (1) this implies (2). □

Corollary 14 in combination with Lemma 8 implies that, for each $z \in V(T)$, it is enough to compute the values $n_z(s)$ for *proper* shapes $s \in \mathsf{shapes}(z)$. To turn this insight into a polynomial time dynamic programming algorithm, we still have to show that the number of proper shapes in $\mathsf{shapes}(z)$ can be polynomially bounded, and that the set of such shapes can be computed in polynomial time. We will achieve this by specifying a subset of $\mathsf{shapes}(z)$ for each $z \in V(T)$ that contains all proper shapes and can be computed in polynomial time.

We define families $\mathscr{X}_z$ and $\overline{\mathscr{X}_z}$ of sets of variables for each node $z \in V(T)$, as follows.

$$\mathscr{X}_z = \{\, X \subseteq var_z : \exists C \in \overline{F_z} \text{ such that } X = var_z \cap var(C) \,\}$$
$$\overline{\mathscr{X}_z} = \{\, X \subseteq \overline{var_z} : \exists C \in F_z \text{ such that } X = \overline{var_z} \cap var(C) \,\}$$

The next lemma follows from the definition of a decomposition tree's index.

Lemma 15. *For every node $z \in V(T)$, $\max(|\mathscr{X}_z|, |\overline{\mathscr{X}_z}|) \le k$.*

Let $z \in V(T)$ and let f be a function with domain $\mathscr{X}_z$ that maps every set X to some projection $f(X) \in \mathsf{proj}(\overline{F_z}|_X, X)$. We denote the set of such functions by $\mathsf{outfunctions}(z)$. Symmetrically, we let $\mathsf{infunctions}(z)$ denote the set of functions g that map every set $Y \in \overline{\mathscr{X}_z}$ to some projection $g(Y) \in \mathsf{proj}(F_z|_Y, Y)$.

Lemma 16. *For every $z \in V(T)$, $|\mathsf{outfunctions}(z)| \le (m+1)^k$ as well as $|\mathsf{infunctions}(z)| \le (m+1)^k$.*

Proof. By Proposition 6 that the cardinality of $\mathsf{proj}(\overline{F_z}|_X, X)$ is bounded by $m+1$ for every $X \in \mathscr{X}_z$. In combination with Lemma 15 this yields $|\mathsf{outfunctions}(z)| \leq (m+1)^k$. The proof of $|\mathsf{infunctions}(z)| \leq (m+1)^k$ is symmetric. □

Let $\mathsf{union}(f)$ denote $\bigcup_{X \in \mathsf{dom}(f)} f(X)$, where $\mathsf{dom}(f)$ is the domain of f. We define the set of *restricted* shapes for $z \in V(T)$ as follows.

$$\begin{aligned}\mathsf{rshapes}(z) = \{\, (out, in) \in \mathsf{shapes}(z) : \exists f \in \mathsf{outfunctions}(z) \; s.t. \; out = \mathsf{union}(f) \\ \wedge \exists g \in \mathsf{infunctions}(z) \; s.t. \; in = \mathsf{union}(g) \,\}\end{aligned}$$

Every pair $(f, g) \in \mathsf{outfunctions}(z) \times \mathsf{infunctions}(z)$ uniquely determines a shape in $\mathsf{rshapes}(z)$. Accordingly, Lemma 16 allows us to bound the cardinality of $\mathsf{rshapes}(z)$ as follows.

Corollary 17. *For any $z \in V(T)$, $|\mathsf{rshapes}(z)| \leq (m+1)^{2k}$.*

Lemma 18. *Let $z \in V(T)$ and let $s \in \mathsf{shapes}(z)$ be proper. Then $s \in \mathsf{rshapes}(z)$.*

This shows that if we can determine the values $n_z(s)$ for every $z \in V(T)$ and $s \in \mathsf{rshapes}(z)$, we can determine the values $n_z(s')$ for every proper shape $s' \in \mathsf{shapes}(z)$. More specifically, as long as we can determine lower bounds for $n_z(s)$ for every $s \in \mathsf{rshapes}(z)$ and the exact values of $n_z(s)$ for proper s, we can compute the correct values for all proper shapes for every tree node.

Definition 19. *For $z \in V(T)$, a* lower bounding function *(for z) associates with each $s \in \mathsf{rshapes}(z)$ a value $l_z(s)$ such that $l_z(s) \leq n_z(s)$ and $l_z(s) = n_z(s)$ if s is proper.*

Let $x, y, z \in V(T)$ such that x and y are the children of z. For each $s \in \mathsf{shapes}(z)$ we write $\mathsf{restricedgen}_z(s) = \mathsf{generators}_z(s) \cap (\mathsf{rshapes}(x) \times \mathsf{rshapes}(y))$.

Lemma 20. *Let $x, y, z \in V(T)$ such that x and y are the children of z. Let l_x and l_y be lower bounding functions for x and y. Let l_z be the function defined as follows. For each $s \in \mathsf{rshapes}(z)$, we let*

$$l_z(s) = \sum_{(s_x, s_y) \in \mathsf{restricedgen}_z(s)} l_x(s_x)\, l_y(s_y). \tag{3}$$

Then l_z is a lower bounding function for z.

Proposition 21. *There is a polynomial p and an algorithm $\mathbb{A}$ such that $\mathbb{A}$, given a CNF formula F and a decomposition tree (T, δ) of $I(F)$, computes the number of satisfying total truth assignments of F in time $m^{6k}p(l)$. Here, m denotes the number of clauses of F, l denotes the length of F, and $k = index(T, \delta)$.*

Proof (Sketch). We compute lower bounding functions for every node of T. It follows from Corollary 17 that each such function can be represented in polynomial space for fixed k. Computing lower bounding functions for leaf nodes is straightforward. For an inner node z with children x and y, we proceed as follows. Assume that lower bounding functions l_x and l_y for x and y have already been computed. We first compute the set $\mathsf{rshapes}(z)$ and set $l_z(s_z) := 0$ for each $s_z \in \mathsf{rshapes}(z)$. We then run through all triples (s_x, s_y, s_z) with $s_x \in \mathsf{rshapes}(x), s_y \in \mathsf{rshapes}(y), s_z \in \mathsf{rshapes}(z)$ and check whether s_x and s_y generate s_z (each check can be done in polynomial time). If that is the case, we set $l_z(s_z) := l_z(s_z) + l_x(s_x)l_y(s_y)$. By Lemma 20, the resulting l_z will be a lower bounding function. There are at most $(m+1)^{6k}$ such triples for each inner node, so this can be done in time $m^{6k}p(l)$ for all nodes of T, where p is a suitable polynomial independent of F. Having computed a lower bounding function l_r for the root r of T, we output $l_r((\emptyset, \emptyset))$, which corresponds to the number of satisfying total truth assignments of F. □

Proof (of Theorem 1). Let $\mathcal{C}$ be a graph class of bounded symmetric clique-width and F a CNF formula of length l with m clauses such that $I(F) \in \mathcal{C}$. Let k be an upper bound for the symmetric clique-width of any graph in $\mathcal{C}$. We compute a decomposition tree (T, δ) of $I(F)$ such that $rankw(T, \delta) = rankw(I(F))$ as follows. Initially, we set $k' := 1$. We then repeatedly run the algorithm of Theorem 5 and increment k' by one until we find a decomposition of rank-width k'. This will be the case after at most k steps since $rankw(I(F)) \leq scw(I(F))$ by Corollary 4. Since $\mathcal{C}$ is fixed, we can consider k (and every $k' \leq k$) a constant, so (T, δ) can be obtained in time $O(|V(I(F))|^3)$ by Theorem 5. Because $2l$ is an upper bound on the number of vertices of $I(F)$, this is in $l^{O(1)}$ (assuming that $l \geq 2$). By Lemma 3, $index(T, \delta) \leq 2^{rankw(I(F))}$ and thus $index(T, \delta) \leq 2^{scw(I(F))} \leq 2^k$. By Proposition 21, the number of satisfying total truth assignments of F can be computed in time $m^{6\, index(T,\delta)}p(l)$ for some polynomial p independent of F, that is, in time $m^{O(2^k)}p(l)$. Since k is a constant, this is in $l^{O(1)}$, as is the total runtime. □

4 Conclusion

We have shown that #SAT is polynomial-time tractable for classes of formulas with incidence graphs of bounded symmetric clique-width (or bounded clique-width, or bounded rank-width). It would be interesting to know whether this problem is tractable under even weaker structural restrictions. For instance, it is currently open whether #SAT is polynomial-time tractable for classes of formulas of bounded β-hypertree width [10] (if a corresponding decomposition is given).

Acknowledgements. The authors would like to thank an anonymous referee for suggesting to state the main results in terms of symmetric clique-width instead of Boolean-width.

References

1. Bacchus, F., Dalmao, S., Pitassi, T.: Algorithms and complexity results for #SAT and Bayesian inference. In: 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), pp. 340–351 (2003)

2. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.: H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. Discrete Applied Mathematics 158(7), 809–819 (2010)
3. Bui-Xuan, B.-M., Telle, J.A., Vatshelle, M.: Boolean-width of graphs. Theoretical Computer Science 412(39), 5187–5204 (2011)
4. Courcelle, B.: Clique-width of countable graphs: a compactness property. Discrete Mathematics 276(1-3), 127–148 (2004)
5. Hliněný, P., Oum, S.I.: Finding branch-decompositions and rank-decompositions. SIAM J. Comput. 38(3), 1012–1032 (2008)
6. Fischer, E., Makowsky, J.A., Ravve, E.R.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discr. Appl. Math. 156(4), 511–529 (2008)
7. Ganian, R., Hliněný, P.: On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. Discr. Appl. Math. 158(7), 851–867 (2010)
8. Ganian, R., Hliněný, P., Obdrzálek, J.: Better algorithms for satisfiability problems for formulas of bounded rank-width. Fund. Inform. 123(1), 59–76 (2013)
9. Gaspers, S., Szeider, S.: Strong backdoors to bounded treewidth SAT. In: Proceedings of FOCS 2013, The 54th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA (to appear, 2013)
10. Gottlob, G., Pichler, R.: Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width. SIAM J. Comput. 33(2), 351–378 (2004)
11. Nishimura, N., Ragde, P., Szeider, S.: Solving #SAT using vertex covers. Acta Informatica 44(7-8), 509–523 (2007)
12. Ordyniak, S., Paulusma, D., Szeider, S.: Satisfiability of acyclic and almost acyclic CNF formulas. Theoretical Computer Science 481, 85–99 (2013)
13. Paulusma, D., Slivovsky, F., Szeider, S.: Model counting for CNF formulas of bounded modular treewidth. In: Portier, N., Wilke, T. (eds.) Proceedings of STACS 2013. LIPIcs, vol. 20, pp. 55–66. Leibniz-Zentrum fuer Informatik (2013)
14. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1-2), 273–302 (1996)
15. Samer, M., Szeider, S.: Algorithms for propositional model counting. J. Discrete Algorithms 8(1), 50–64 (2010)
16. Sang, T., Beame, P., Kautz, H.A.: Performing Bayesian inference by weighted model counting. In: Proceedings of the 20th National Conference on Artificial Intelligence, AAAI 2005, vol. 1, pp. 475–481. AAAI Press (2005)
17. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 188–202. Springer, Heidelberg (2004)
18. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)

Computing Plurality Points and Condorcet Points in Euclidean Space

Yen-Wei Wu[1], Wei-Yin Lin[1], Hung-Lung Wang[4], and Kun-Mao Chao[1,2,3]

[1] Department of Computer Science and Information Engineering
[2] Graduate Institute of Biomedical Electronics and Bioinformatics
[3] Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106
[4] Institute of Information and Decision Sciences
National Taipei College of Business, Taipei, Taiwan 100

Abstract. This work concerns two kinds of spatial equilibria. Given a multiset of n points in Euclidean space equipped with the ℓ_2-norm, we call a location a *plurality point* if it is closer to at least as many given points as any other location. A location is called a *Condorcet point* if there exists no other location which is closer to an absolute majority of the given points. In d-dimensional Euclidean space $\mathbb{R}^d$, we show that the plurality points and the Condorcet points are equivalent. When the given points are not collinear, the Condorcet point (which is also the plurality point) is unique in $\mathbb{R}^d$ if such a point exists. To the best of our knowledge, no efficient algorithm has been proposed for finding the point if the dimension is higher than one. In this paper, we present an $O(n^{d-1} \log n)$-time algorithm for any fixed dimension $d \geq 2$.

1 Introduction

In this paper, we search for the most "popular" facility locations in Euclidean space regarding the voting system, in which the spatial equilibrium is measured by two election criteria in the voting theory, the plurality [2, 10, 18] and the Condorcet winner [2,9,10]. We are given a multiset of points, regarded as *voters*, in Euclidean space equipped with the ℓ_2-norm. A *plurality point* is a location which is closer to at least as many voters as any other location [2]. A *Condorcet point* is a location such that there is no other location closer to a strict majority of voters, i.e., no other location is closer to more than a half of the voters than the Condorcet point.

1.1 Motivation

Determining the best locations to place facilities by voting is essential in the construction of public infrastructures. Another important application arises in the process of election. The electoral process can be reduced to a multidimensional model of spatial competition [6, 19], i.e., the political spectrum. In the one-dimensional case, each side of the real line represents the left or right wing with respect to a certain issue. Suppose that there are several citizens and two issues: one is environmental and the other is economic. Each issue corresponds to a dimension of $\mathbb{R}^2$. The coordinate of each citizen is his or her most preferred position in the issue space, where their preferences of both issue are quantified. To help analyzing the trend of events, the equilibrium can be found by computing plurality points or Condorcet points. The model can be extended to $\mathbb{R}^d$ for multidimensional situations when there are even more issues.

Most previous works appeared mainly in the literature of economics, politics and operations research [9,13,19]. Although some good properties of these spatial equilibria have been discovered, they are not specific for algorithmic usage. To the best of our knowledge, no efficient algorithm has been proposed in Euclidean space.

1.2 Related Work

Facility location problems in computational geometry have been studied for decades. Motivated by different applications, various measurements are considered. Among those interesting criteria with different applications, searching for a point in the plane which minimizes the sum of distances to a given set of points is one of the most well-investigated problems; such a point is often known as the *geometric median* or the *Fermat-Weber point* [11].

Under certain circumstances, Condorcet points and median points coincide. The nature of this property is intuitive on a real line. Chepoi and Dragan [4] showed that Condorcet and median points of a simple rectilinear polygon coincide, and the set can be found in $O(N + n \log N)$ time, where N is the number of vertices and n is the number of voters. Bandelt [2] characterized the conditions where Condorcet points and Fermat-Weber points coincide on a graph and gave a polynomial-time algorithm which decides whether a given graph has Condorcet points. The comparison between Condorcet points and Fermat-Weber points was discussed by Hansen and Thisse [9], and a polynomial-time algorithm to determine the set of Condorcet points of a graph was provided in [8]. Condorcet points on a graph were also investigated by Labbé [12] and Vohra [17].

The criterion of plurality was proposed mainly in the theory of spatial competition [19]. By considering it as the equilibrium point of social decisions under majority rule, some researchers proposed new perspectives in two-dimensional space, as well as on a graph [18, 19]. There were also some prior results discussing the relationship between plurality points and Condorcet points on different normed spaces [7, 13].

The rest of this paper is organized as follows. First, we formally define the problems and summarize some basic properties of plurality and Condorcet points in Section 2. The equivalence of the two criteria in Euclidean space is also shown. In Section 3, we give an $O(n \log n)$-time algorithm to compute plurality points in $\mathbb{R}^2$. In Sections 4, we further show how to compute the point in higher-dimensional space by extending the result in $\mathbb{R}^2$ to $\mathbb{R}^d$. Finally, Section 5 concludes this paper by summarizing the main results and future work. Due to the space limitation, some proofs and the algorithm for the higher-dimensional case are omitted and will be given in the journal version.

2 Preliminaries

2.1 Problem Definition

Suppose that there is a multiset of *voters* $V = \{v_1, v_2, \ldots, v_n\}$, where v_i is a point in $\mathbb{R}^d$. We assume that n is finite. For each voter v_i, we let $(v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(d)})$ denote the coordinate of v_i. The distance $d(u, v)$ between two points u and v in $\mathbb{R}^d$ is measured by the ℓ_2-norm, i.e. $d(u,v) = \sqrt{\sum_{i=1}^{d}(u^{(i)} - v^{(i)})^2}$. As in [19], a point θ is said to be a *multidimensional median* of V if $|\{v \in V : v^{(j)} \leq \theta^{(j)}\}| \geq n/2$ and $|\{v \in V : v^{(j)} \geq \theta^{(j)}\}| \geq n/2$, for all j such that $1 \leq j \leq d$. Let $\mathcal{M}$ be the set of all multidimensional medians of V.

For two arbitrary points $\theta_1 \in \mathbb{R}^d$ and $\theta_2 \in \mathbb{R}^d$, we say that voter v_i *prefers* θ_1 to θ_2 if $d(v_i, \theta_1) < d(v_i, \theta_2)$. Define $[\theta_1 \succ \theta_2] = \{v_i \in V : d(v_i, \theta_1) < d(v_i, \theta_2)\}$ and $[\theta_1 \sim \theta_2] = V \setminus ([\theta_1 \succ \theta_2] \cup [\theta_2 \succ \theta_1])$. A point $\theta \in \mathbb{R}^d$ is called a plurality point if and only if $|[\theta^* \succ \theta]| \leq |[\theta \succ \theta^*]|$ holds for each $\theta^* \in \mathbb{R}^d$. A point $\theta \in \mathbb{R}^d$ is called a Condorcet point if and only if $|[\theta^* \succ \theta]| \leq n/2$ holds for each $\theta^* \in \mathbb{R}^d$. Formal definitions of the problems are given below.

Problem 1 (The d-Dimensional Plurality-Points Problem). Given a multiset V consisting of n points in $\mathbb{R}^d$, the d-dimensional plurality-points problem is to find the set of all plurality points of V.

Problem 2 (The d-Dimensional Condorcet-Points Problem). Given a multiset V consisting of n points in $\mathbb{R}^d$, the d-dimensional Condorcet-points problem is to find the set of all Condorcet points of V.

2.2 The Equivalence of Plurality Points and Condorcet Points in $\mathbb{R}^d$

Both plurality points and Condorcet points are often used to represent the equilibria in a space under majority rules. Relationships between plurality points and Condorcet points under different assumptions were discussed in [2, 13]. In Lemma 1, we show that a plurality point is a Condorcet point in $\mathbb{R}^d$ with ℓ_2-norm metric, and vice versa.

Lemma 1. *In $\mathbb{R}^d$, θ is a plurality point if and only if θ is a Condorcet point.*

Proof. By definition, a plurality point is a Condorcet point. It remains to show that a Condorcet point is a plurality point. Suppose on the contrary that x is a Condorcet point but not a plurality point. Since x is not a plurality point, there is a point y satisfying $|[x \succ y]| < |[y \succ x]|$. Let z be the midpoint of $\overline{xy}$. For each $c \in [y \succ x] \cup [x \sim y]$, $d(c, z) < d(c, x)$. Thus, $|[z \succ x]| > n/2$, which is a contradiction. $\square$

Since we have shown the equivalence of plurality points and Condorcet points in Euclidean space, the answers of Problem 1 and Problem 2 must be the same. In the rest of this paper, a solution to Problem 1 will be introduced, and it can be used to solve Problem 2 as well.

3 The Two-Dimensional Plurality-Points Problem

Let us first consider the special case where all voters are collinear. We denote $\triangle_V$ as the set consisting of all the plurality points of voters V.

Lemma 2. *Let V be a multiset of voters in $\mathbb{R}^2$. If all voters in V are collinear, then*

(i) $\triangle_V = \mathcal{M}$;
(ii) $|\triangle_V| \geq 1$.

Proof. For (i), it has been shown that $\mathcal{M}$ equals the set of all Condorcet points in $\mathbb{R}^2$ when all voters are collinear [9]. Together with Lemma 1, $\triangle_V = \mathcal{M}$.

For (ii), by the definition of a multidimensional median, $\mathcal{M} \neq \emptyset$. Notice that $|\mathcal{M}| > 1$ may happen when $|V|$ is even. Thus, $|\triangle_V| = |\mathcal{M}| \geq 1$. □

Because $\triangle_V = \mathcal{M}$ when all voters are collinear, methods for finding the multidimensional medians of collinear voters can be used to solve the special case of Problem 1 and will be introduced later in this section. The non-collinear case is handled as follows. The two-dimensional space is partitioned by any given line L into $L^+ \cup L^- \cup L$, where L^+ and L^- are the sets of points belonging to different sides of L, respectively. Formally, let L be a line with equation $y = mx + b$, where m is the slope and b is the y-intercept. We define $L^+ = \{x \in \mathbb{R}^2 : mx^{(1)} - x^{(2)} + b > 0\}$, $L^- = \{x \in \mathbb{R}^2 : mx^{(1)} - x^{(2)} + b < 0\}$, and $L = \{x \in \mathbb{R}^2 : mx^{(1)} - x^{(2)} + b = 0\}$. The sets $V \cap L^+$, $V \cap L^-$, and $V \cap L$ are denoted by V_L^+, V_L^-, and V_L, and their cardinalities are denoted by n_L^+, n_L^-, and n_L, respectively. Define $V_{L,\triangle}^{\nearrow} = \{v \in V_L : v^{(1)} > \triangle^{(1)}\} \cup \{v \in V_L : v^{(1)} = \triangle^{(1)} \text{ and } v^{(2)} > \triangle^{(2)}\}$, and $V_{L,\triangle}^{\swarrow} = \{v \in V_L : v^{(1)} < \triangle^{(1)}\} \cup \{v \in V_L : v^{(1)} = \triangle^{(1)} \text{ and } v^{(2)} < \triangle^{(2)}\}$. Besides, let $n_{L,\triangle}^{\nearrow}$ and $n_{L,\triangle}^{\swarrow}$ be the cardinalities of $V_{L,\triangle}^{\nearrow}$ and $V_{L,\triangle}^{\swarrow}$, respectively. In addition, the function $d_L : \mathbb{R}^2 \to \mathbb{R}$ is defined to be the distance from a point to the line L.

Lemma 3. *In $\mathbb{R}^2$, $\triangle$ is a plurality point if and only if for any line L passing through $\triangle$, $n_L^+ \leq n/2$ and $n_L^- \leq n/2$.*

Proof. For necessity, we prove by contradiction. Without loss of generality, we assume that there is a line L passing through $\triangle$ such that $n_L^+ > n/2$. Let $L_\parallel$ be a line passing through v_+ parallel to L, where $v_+ \in \arg\min_{v \in V_L^+} d_L(v)$, and let $L_\perp$ be a line passing through $\triangle$ perpendicular to L. Pick $\triangle'$ as the intersection of $L_\parallel$ and $L_\perp$. Clearly, for each point v in V_L^+, v prefers $\triangle'$ to $\triangle$. Since $n_L^+ > n/2$, it follows that $|[\triangle \succ \triangle']| < |[\triangle' \succ \triangle]|$, which contradicts that $\triangle$ is a plurality point.

For sufficiency, let $\triangle'$ be an arbitrary point other than $\triangle$, and let L be the line passing through both $\triangle'$ and $\triangle$. For the line $L_\perp$ perpendicular to L at $\triangle$, we may assume without loss of generality that $\triangle' \in L_\perp^+$. Evidently, for each voter v in $V_{L_\perp}^- \cup V_{L_\perp}$, v prefers $\triangle$ to $\triangle'$. Because of the assumption that $n_{L_\perp}^+ \leq n/2$, we have that $n/2 \leq n_{L_\perp}^- + n_{L_\perp}$, and thus $|[\triangle' \succ \triangle]| \leq |[\triangle \succ \triangle']|$. □

The equivalence condition given in Lemma 3 can be applied to verify if a given point $\triangle$ is a plurality point. Notice that all lines passing through $\triangle$ have to be examined, even for those containing no voter. Let $\mathcal{L}_\triangle$ be the set of lines passing through $\triangle$. For a line $L \in \mathcal{L}_\triangle$, if $n_L^+ \leq n/2$ and $n_L^- \leq n/2$, then each of the two closed halfspaces separated by L contains at least $n/2$ voters, and vice versa. Thus, we have that

$$\min_{L \in \mathcal{L}_\triangle} \{|V \cap \gamma| : \gamma \text{ is a closed halfspace separated by } L\} \geq n/2.$$

The number $\min_{L \in \mathcal{L}_\triangle} \{|V \cap \gamma| : \gamma \text{ is a closed halfspace separated by } L\}$ is defined as the *Tukey depth* of $\triangle$ relative to V [16]. To compute the Tukey depth of a point relative to n points in $\mathbb{R}^2$, an $O(n \log n)$-time algorithm was proposed by Rousseeuw and Struyf [15]. Thus, in $\mathbb{R}^2$ one may verify if a given point is a plurality point in $O(n \log n)$ time. For any set of points S, the set $S \cap \triangle_V$ can be computed by iteratively applying the above steps, and the procedure is denoted as VerifyCandidates$(S, V, 2)$, where the third cell stands for the dimension of the space. The procedure is of time complexity $O(|S| \cdot n \log n)$.

Lemma 4. *Let V be a multiset of voters in $\mathbb{R}^2$. If not all the voters are collinear, then*

(i) $\triangle_V \subseteq \mathcal{M}$;
(ii) either $|\triangle_V| = 1$ or $|\triangle_V| = 0$.

Proof. For (i), suppose that $\triangle$ is a plurality point. Let L_1 and L_2 be two lines passing through $\triangle$ parallel to the y-axis and the x-axis, respectively. By Lemma 3, $n/2 \leq n_{L_1}^+$, $n/2 \leq n_{L_1}^-$, $n/2 \leq n_{L_2}^+$, and $n/2 \leq n_{L_2}^-$, and thus $n_{L_1}^+ + n_{L_1} \geq n/2$, $n_{L_1}^- + n_{L_1} \geq n/2$, $n_{L_2}^+ + n_{L_2} \geq n/2$, and $n_{L_2}^- + n_{L_2} \geq n/2$. These entail that $\triangle$ is a multidimensional median.

For (ii), suppose on the contrary that $|\triangle_V| \geq 2$, and $\triangle_1$ and $\triangle_2$ are two distinct plurality points. Let a be the midpoint of $\overline{\triangle_1 \triangle_2}$, and let $v \in V$ be a point that is not on $\overleftrightarrow{\triangle_1 \triangle_2}$. Let L_1 and L_2 be two lines parallel to $\overleftrightarrow{va}$ through $\triangle_1$ and $\triangle_2$, respectively. Without loss of generality, assume that $\triangle_2 \in L_1^+$. It follows that $c \in L_1^+$ for each voter $c \in L_2^+ \cup L_2 \cup \{v\}$. Therefore,

$$n/2 \leq n_{L_2}^+ + n_{L_2} < n_{L_2}^+ + n_{L_2} + 1 = |L_2^+ \cup L_2 \cup \{v\}| \leq n_{L_1}^+,$$

where the first inequality follows from Lemma 3. Again, by Lemma 3, there is a contradiction that $\triangle_1$ is a plurality point. □

Lemma 4 demonstrates that a plurality point must be a multidimensional median, but a multidimensional median may not be a plurality point. For example, if not all the voters are collinear and n is even, the number of multidimensional medians can be infinite, yet the number of plurality points is at most one. In such circumstances, there must be a multidimensional median that is not a plurality point. Moreover, notice that the case where $|\triangle_V| = 0$ may happen if not all the voters are collinear. An example is given in Figure 1. Let $V = \{v_1, v_2, v_3\}$, and the voters are not collinear. For each point θ_1 outside triangle $v_1v_2v_3$, there is always an edge e passing through two voters satisfying that θ_1 and the third voter are on the different sides of e. Draw a line passing through θ_1 perpendicular to e at a point θ_3. It can be derived that all three voters prefer θ_3 to θ_1. Similarly, for each point θ_2 inside triangle $v_1v_2v_3$, there is a point θ_3 preferred by at least two voters. Lastly, for each point θ on an edge of triangle $v_1v_2v_3$, let e be an edge which does not go through θ, and let θ_4 be a perpendicular point of $\overleftrightarrow{\theta\theta_4}$ on e. Apparently, θ_4 is preferred by at least two voters.

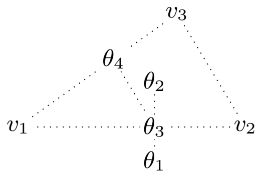


Fig. 1. An illustration showing that a plurality point may not exist

In addition, Lemma 4 gives two necessary conditions of a plurality point when the voters are not collinear. It indicates that one of the multidimensional medians may be the unique plurality point. Nevertheless, the number of multidimensional medians may still be infinite. To futher shrink the search space, we need more clues. In the following four lemmas, properties of two different cases where $\triangle \in V$ and $\triangle \notin V$ are discussed separately. With these properties, it suffices to verify only $O(1)$ points in $\mathbb{R}^2$.

Let $V^{(i)} = \{v_1^{(i)}, v_2^{(i)}, \ldots, v_n^{(i)}\}$, for $i = 1, 2, \ldots, d$. Denote $x_h^{(i)}$ and $x_l^{(i)}$ as the $\lceil (n+1)/2 \rceil$-th and $\lceil n/2 \rceil$-th smallest numbers in $V^{(i)}$. It is not difficult to show that for each multidimensional median $q \in \mathcal{M}$, $x_l^{(i)} \leq q^{(i)} \leq x_h^{(i)}$. We further denote the set $\mathcal{C} = \{(x_{t_1}^{(1)}, x_{t_2}^{(2)}, \ldots, x_{t_d}^{(d)}) : t_1, t_2, \ldots, t_d \in \{l, h\}\}$, which collects all the *corners* of $\mathcal{M}$. Clearly, $|\mathcal{C}| \leq 2^d$.

Lemma 5. *Let V be a multiset of voters in $\mathbb{R}^d$ and $\triangle$ be a plurality point of V. If $\triangle \in V$, then $\triangle \in \mathcal{C}$, and $|V \cap \mathcal{C}| \leq 2^d$.*

Proof. By Lemma 2 and Lemma 4, $\triangle \in \triangle_V \subseteq \mathcal{M}$. According to the definition of $\mathcal{C}$, $V \cap \mathcal{M} \subseteq \mathcal{C}$. Therefore, we have that $\triangle \in \mathcal{C}$ if $\triangle \in V$. Since $|\mathcal{C}| \leq 2^d$, $|V \cap \mathcal{C}| \leq 2^d$ holds. □

Lemma 6. *Let V be a multiset of voters and $\triangle$ be a point in $\mathbb{R}^2$. If $\triangle \notin V$, then the following statements are equivalent:*

(i) $\triangle$ is a plurality point;
(ii) $n_L^+ = n_L^-$ for each line L passing through $\triangle$.

Proof. We prove the equivalence by showing that (ii)⇒(i) and (i)⇒(ii).

- (ii)⇒ (i):
  Since for each line L passing through $\triangle$, $n_L^+ = n_L^-$ and $n_L^+ + n_L^- \leq n$, by Lemma 3, $\triangle$ is a plurality point.
- (i)⇒ (ii):
  Let L be a line passing through $\triangle$. If $V_L = \emptyset$, $n_L^+ + n_L^- = n$. Together with Lemma 3, we have that $n_L^+ = n_L^- = n/2$. Otherwise, let m be the slope of L. Because of its similarity, we consider only the case where $m > 0$ and $m < \infty$. Let L_1 and L_2 be two lines resulting from slightly rotating L around $\triangle$ such that the slopes of L_1 and L_2 are positive and
  - $V_{L_1} = \emptyset$, $V_{L_1}^+ = V_L^+ \cup V_{L,\triangle}^{\swarrow}$, and $V_{L_1}^- = V_L^- \cup V_{L,\triangle}^{\nearrow}$.
  - $V_{L_2} = \emptyset$, $V_{L_2}^+ = V_L^+ \cup V_{L,\triangle}^{\nearrow}$, and $V_{L_2}^- = V_L^- \cup V_{L,\triangle}^{\swarrow}$.

  L_1 and L_2 must exist because n is finite and $\triangle\notin V$. Since $V_{L_1} = \emptyset$, $n_{L_1}^+ + n_{L_1}^- = n$. Together with Lemma 3, we have that

$$n_L^- + n_{L,\triangle}^{\nearrow} = n_{L_1}^- = n/2 = n_{L_1}^+ = n_L^+ + n_{L,\triangle}^{\swarrow}. \tag{1}$$

  Similarly,

$$n_L^- + n_{L,\triangle}^{\swarrow} = n_{L_2}^- = n/2 = n_{L_2}^+ = n_L^+ + n_{L,\triangle}^{\nearrow}. \tag{2}$$

  By (1)+(2), we have that $n_L^- = n_L^+$ and $n_{L,\triangle}^{\nearrow} = n_{L,\triangle}^{\swarrow}$.

□

For the case where $\triangle\notin V$, Lemma 7 and Lemma 8 can be utilized to dramatically reduce the cardinality of the search space.

Lemma 7. *Let V be a multiset of voters. If $\triangle\notin V$ is a plurality point and $|\mathcal{C}| > 1$, then $\triangle\notin \mathcal{C}$.*

Lemma 8. *Let V be a multiset of voters in $\mathbb{R}^2$, $\triangle$ be a plurality point of V, and L be a line passing through $\triangle$ with $n_L = 0$. If not all voters are collinear and $\triangle\notin V$, then $\triangle$ is the intersection of the common internal tangent lines of $\mathcal{H}_a$ and $\mathcal{H}_b$, where $\mathcal{H}_a$ and $\mathcal{H}_b$ are the convex hulls of V_L^+ and V_L^-, respectively.*

Proof. V_L^+ and V_L^- are disjoint, and they have a separating line L. Since not all voters are collinear and $\triangle\notin V$ is a plurality point, we have that $n \geq 4$. Together with Lemma 6 and that $n_L = 0$, $n_L^+ = n_L^- = n/2 \geq 2$. Therefore, the intersection $\triangle'$ of the common internal tangent lines exists and is unique. Suppose on the contrary that $\triangle\neq\triangle'$. Let L_1 be a common internal tangent line which does not include $\triangle$, and L_2 be a line passing through $\triangle$ parallel to L_1. Without loss of generality, assume that $V_L^+ \subseteq V_{L_1} \cup V_{L_1}^+$ and $V_{L_1} \subseteq V_{L_2}^+$. It follows that $V_L^+ \subseteq V_{L_1} \cup V_{L_1}^+ \subseteq V_{L_2}^+$. Since L_1 is a common internal tangent line of $\mathcal{H}_a$ and $\mathcal{H}_b$, there exists a tangent point v such that $v \in V_L^-$ and $v \in V_{L_2}^+$. Thus, we have $n_{L_2}^+ \geq n_{L_1} + n_{L_1}^+ > n/2$, which contradicts the assumption that $\triangle$ is a plurality point according to Lemma 3. □

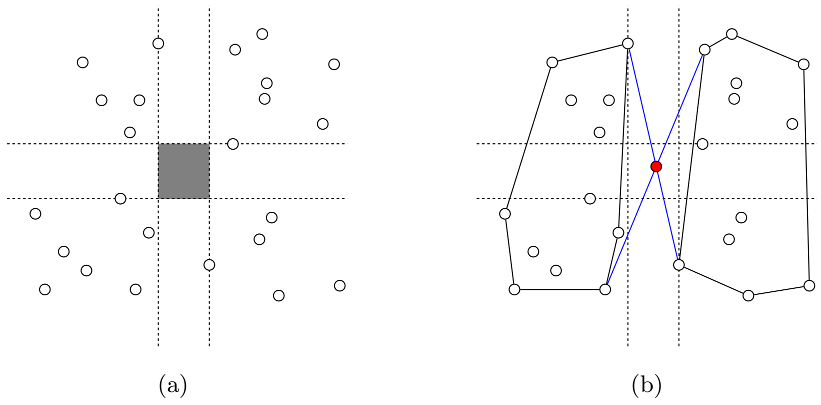


Fig. 2. Reducing the cardinality of the search space from ∞ to 1, where $\triangle \notin V$. (a) The hollow points are voters, and the shaded region is the search space, which is $\mathcal{M}$. (b) The solid point is the intersection of the two common internal tangent lines of two convex hulls. It is the only candidate for being a plurality point.

An illustration of Lemma 8 is given in Figure 2. We now give an algorithm, named PluralityPoint2D, for solving the two-dimensional plurality-points problem. This algorithm computes the set of all plurality points. Its pseudocode is given in Algorithm 1.

By Lemma 2, if all voters are collinear, lines 1–2 compute the plurality points. Otherwise, we compute the unique plurality point according to the cardinality of the set of the corners of multidimensional medians of V, which is determined in lines 4–8. If $|\mathcal{C}| = 1$, $\mathcal{M} = \mathcal{C}$. By Lemma 4, the single point in $\mathcal{C}$ is the only candidate that can be the plurality point and is tested by VerifyCandidates (line 10). For $|\mathcal{C}| > 1$, if a plurality point $\triangle$ exists, either $\triangle \in V$ or $\triangle \notin V$. If $\triangle \in V$, by Lemma 5, $\triangle \in V \cap \mathcal{C}$, and $\triangle$ is returned in line 12. If $\triangle \notin V$, $\triangle \in \mathcal{M} - \mathcal{C}$ according to Lemma 4 and Lemma 7. Since $\triangle \in \mathcal{M} - \mathcal{C}$ and $\triangle \notin V$, there is always a line L passing through $\triangle$ parallel to one of the axes with $n_L = 0$. Lines 14–19 compute V_L^+ and V_L^-. By Lemma 8, the plurality point is the intersection of the common internal tangents of the convex hulls of V_a and V_b. The intersection is obtained in lines 20–21, and is lastly verified in line 22.

Theorem 1. *The two-dimensional plurality-points problem can be solved in $O(n \log n)$ time.*

Proof. The correctness of PluralityPoint2D has been justified above, and it remains to show that PluralityPoint2D runs in $O(n \log n)$ time.

By going through all voters in V and checking all the slopes of $\overline{v_1 v_i}$ where $2 \leq i \leq n$, line 1 can be done in linear time. By using the famous selection algorithm in [5], it takes linear time to finish line 2 and lines 4 to 8. Because $|\mathcal{C}|$ and $|V \cap \mathcal{C}|$ are both constant, the running time of lines 9 to 12 is $O(n \log n)$. Since CommonInternalTangents(V_a, V_b) can be done by linear programming in $\mathbb{R}^2$, the running time is $O(n)$ [14], and thus lines 13 to 22 takes $O(n)$ time.

Algorithm 1. PLURALITYPOINT2D($V = \{v_1, v_2, \ldots, v_n\}$)

```
1  if all voters are collinear then
2      $P$ := the multidimensional medians of $V$;
3  else
       /* SELECT($S$,$k$) returns the $k$-th smallest number in $S$. */
4      $x_h$ :=SELECT($V^{(1)}$, $\lceil (n+1)/2 \rceil$);
5      $x_l$ :=SELECT($V^{(1)}$, $\lceil n/2 \rceil$);
6      $y_h$ :=SELECT($V^{(2)}$, $\lceil (n+1)/2 \rceil$);
7      $y_l$ :=SELECT($V^{(2)}$, $\lceil n/2 \rceil$);
8      $\mathcal{C}$:=$\{(x_h, y_h), (x_h, y_l), (x_l, y_h), (x_l, y_l)\}$;
9      if $|\mathcal{C}| = 1$ then
10         $P$ := VERIFYCANDIDATES($\mathcal{C}$);
11     else
           /* the case where $\triangle \in V$ */
12         $P$ := VERIFYCANDIDATES($V \cap \mathcal{C}$);
13         if $P = \emptyset$ then
               /* the case where $\triangle \notin V$ */
14             if $x_h = x_l$ then
15                 $V_a$:=$\{v \in V : v^{(2)} \geq y_h\}$;
16                 $V_b$:=$\{v \in V : v^{(2)} \leq y_l\}$;
17             else
18                 $V_a$:=$\{v \in V : v^{(1)} \geq x_h\}$;
19                 $V_b$:=$\{v \in V : v^{(1)} \leq x_l\}$;
20             $(L_a, L_b)$:= COMMONINTERNALTANGENTS($V_a, V_b$);
21             $p$ := the intersection of $L_a$ and $L_b$;
22             $P$ := VERIFYCANDIDATES($\{p\}$);
23 return $P$;
```

To sum up, the two-dimensional plurality-points problem can be solved by PLURALITYPOINT2D in $O(n \log n)$ time. □

4 The Higher-Dimensional Plurality-Points Problem

In this section, we consider the plurality-points problem with $d \geq 3$. Lemmas 3, 4, 6, 7, and 8 are extended to Lemmas 9, 10, 11, 12, and 13. Based on these lemmas, we propose the algorithm PLURALITYPOINT for the higher-dimensional case.

The distance from a point p to a subspace $S \subseteq \mathbb{R}^d$ is denoted by $d_S(p)$. Given a hyperplane E, the space $\mathbb{R}^d$ is partitioned into $E^+ \cup E^- \cup E$, where E^+ and E^- are two open halfspaces separated by E. Let $V_E^+ = V \cap E^+$, $V_E^- = V \cap E^-$, and $V_E = V \cap E$. We denote their cardinalities by n_E^+, n_E^-, and n_E, respectively.

Lemma 9. *In $\mathbb{R}^d$, $\triangle$ is a plurality point if and only if for any hyperplane E passing through $\triangle$, $n_E^+ \leq n/2$ and $n_E^- \leq n/2$.*

Lemma 10. *Let V be a multiset of voters in $\mathbb{R}^d$. If not all the voters are collinear, then*

(i) $\triangle_V \subseteq \mathcal{M}$;
(ii) either $|\triangle_V| = 1$ *or* $|\triangle_V| = 0$.

Lemma 11. *Let V be a multiset of voters and $\triangle$ be a point in $\mathbb{R}^d$. If $\triangle \notin V$, then the following statements are equivalent:*

(i) $\triangle$ is a plurality point;
(ii) $n_E^+ = n_E^-$ for each hyperplane E passing through $\triangle$.

Lemma 12. *Let V be a multiset of voters in $\mathbb{R}^d$. If $\triangle \notin V$ is a plurality point and $|\mathcal{C}| > 1$, then $\triangle \notin \mathcal{C}$.*

Lemma 13. *Let V be a multiset of voters in $\mathbb{R}^d$, $\triangle$ be a plurality point of V, and E be a hyperplane passing through $\triangle$ with $n_E = 0$. If not all voters are coplanar and $\triangle \notin V$, then $\triangle$ is the intersection of d common internal tangent hyperplanes of $\mathcal{H}_a$ and $\mathcal{H}_b$, where $\mathcal{H}_a$ and $\mathcal{H}_b$ are the convex hulls of V_E^+ and V_E^-, respectively.*

Theorem 2. *The plurality-points problem in $\mathbb{R}^d$ can be solved in $O(n^{d-1} \log n)$ time for any fixed d.*

5 Concluding Remarks

In Euclidean space, we show that plurality points and Condorcet points are equivalent. We provide an $O(n^{d-1} \log n)$-time algorithm to find them in $\mathbb{R}^d$ for any fixed dimension d. All steps take $O(n)$ time except VerifyCandidate.

We note here that a plurality point in Euclidean space is precisely a *Tukey median* with depth $\lceil n/2 \rceil$, where a Tukey median is a point with the largest Tukey depth. The fact is derived since the largest Tukey depth is, by definition, upper bounded by $\lceil n/2 \rceil$, and by Lemma 9, we have that the Tukey depth of a plurality point is greater than or equal to $n/2$. As a result, computing a plurality point can be reduced to computing a Tukey median and determining if its Tukey depth is $\lceil n/2 \rceil$. Nevertheless, most results on computing the Tukey median are in low dimensions, as surveyed in [3], and the current best time complexities are asymptotically worse than those of the algorithms proposed in this paper (e.g., $O(n \log^3 n)$ to $O(n \log n)$ in the 2-dimensional space, and $O(n^2 \text{polylog } n)$ to $O(n^2 \log n)$ in the 3-dimensional space). For higher-dimensional cases, only randomized algorithms have been proposed for computing the Tukey median [1, 3]. It remains open to clarify the gap between computing a Tukey median and a plurality point.

Acknowledgements. Yen-Wei Wu, Wei-Yin Lin, and Kun-Mao Chao were supported in part by NSC grants 100-2221-E-002-131-MY3 and NSC 101-2221-E-002-063-MY3, and Hung-Lung Wang was supported in part by NSC grant 101-2221-E-141-002- from the National Science Council, Taiwan.

References

1. Afshani, P.: On approximate range counting and depth. Discrete and Computational Geometry 42(1), 3–21 (2009)
2. Bandelt, H.-J.: Networks with Condorcet solutions. European Journal of Operational Research 20(3), 314–326 (1985)
3. Chan, T.M.: An optimal randomized algorithm for maximum Tukey depth. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 430–436 (2004)
4. Chepoi, V., Dragan, F.: Condorcet and median points of simple rectilinear polygons. Location Science 4(1-2), 21–35 (1996)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
6. Davis, O.A., Hinich, M.J., Ordeshook, P.C.: An expository development of a mathematical model of the electoral process. The American Political Science Review 64(2), 426–448 (1970)
7. Durier, R.: Continuous location theory under majority rule. Mathematics of Operations Research 14(2), 258–274 (1989)
8. Hansen, P., Labbé, M.: Algorithms for voting and competitive location on a network. Transportation Science 22(4), 278–288 (1988)
9. Hansen, P., Thisse, J.-F.: Outcomes of voting and planning: Condorcet, Weber and Rawls locations. Journal of Public Economics 16(1), 1–15 (1981)
10. Hansen, P., Thisse, J.-F., Wendell, R.E.: Equivalence of solutions to network location problems. Mathematics of Operations Research 11(4), 672–678 (1986)
11. Kuhn, H.W.: On a pair of dual nonlinear programs. In: Abadie, J. (ed.) Nonlinear Programming, pp. 37–54. North-Holland, Amsterdam (1967)
12. Labbé, M.: Outcomes of voting and planning in single facility location problems. European Journal of Operational Research 20(3), 299–313 (1985)
13. McKelvey, R.D., Wendell, R.E.: Voting equilibria in multidimensional choice spaces. Mathematics of Operations Research 1(2), 144–158 (1976)
14. Megiddo, N.: Linear programming in linear time when the dimension is fixed. Journal of the ACM 31(1), 114–127 (1984)
15. Rousseeuw, P.J., Struyf, A.: Computing location depth and regression depth in higher dimensions. Statistics and Computing 8(3), 193–203 (1998)
16. Tukey, J.: Mathematics and the picturing of data. In: Proceedings of the International Congress of Mathematicians, vol. 2, pp. 523–531 (1975)
17. Vohra, R.V.: Distance weighted voting and a single facility location problem. European Journal of Operational Research 41(3), 314–320 (1989)
18. Wendell, R.E., McKelvey, R.D.: New perspectives in competitive location theory. European Journal of Operational Research 6(2), 174–182 (1981)
19. Wendell, R.E., Thorson, S.J.: Some generalizations of social decisions under majority rule. Econometrica 42(5), 893–912 (1974)

Computing Minimum Tile Sets to Self-Assemble Color Patterns*

Aleck C. Johnsen[1], Ming-Yang Kao[1], and Shinnosuke Seki[2]

[1] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, USA
aleckjohnsen2012@u.northwestern.edu, kao@northwestern.edu
[2] Department of Information and Computer Science, Aalto University, P. O. Box 15400, FI-00076, Aalto, Finland
shinnosuke.seki@aalto.fi

Abstract. Patterned self-assembly tile set synthesis (PATS) aims at finding a minimum tile set to uniquely self-assemble a given rectangular pattern. For $k \geq 1$, k-PATS is a variant of PATS that restricts input patterns to those with at most k colors. We prove the $\mathcal{NP}$-hardness of 29-PATS, where the best known is that of 60-PATS.

1 Introduction

Pattern painting is one of the main goals of research on DNA self-assembly. Various patterns of practical significance were exhibited to self-assemble from DNA tiles, including the Sierpinski triangle [6] and binary counter [2]. As illustrated in Fig. 1, tiles of blue tile types `B1`, `B2` and red tile types `R1`, `R2` implement a half-subtractor and self-assemble a reverse binary counter.

Patterned Self-Assembly Tile Set Synthesis. (PATS) [4] aims at finding a minimum tile type set with which a rectilinear tile assembly system uniquely self-assembles a given pattern. For $k \geq 1$, **k-PATS** is a PATS subproblem of practical significance in which inputs are restricted to patterns with at most k colors. Seki proved that 60-PATS is $\mathcal{NP}$-hard [8]. He designed a set T of 84 tile types such that for an evaluator circuit pattern $P(\phi)$ reduced from a given 3SAT instance ϕ, T self-assembles $P(\phi)$ from a seed if and only if ϕ is satisfiable. Subpatterns of $P(\phi)$ imply that any set with fewer tile types than T can not assemble $P(\phi)$, for any ϕ. Tile types in T implement `OR`, `AND`, and `NOT` gates to evaluate ϕ, and the unique minimality of T for $P(\phi)$ required 60 colors.

In this paper we employ SUBSET SUM rather than 3SAT, which can be evaluated using the half-subtractor of Brun [3], and prove a stronger result.

Theorem 1. *29-PATS is $\mathcal{NP}$-hard.*

* This work is supported in part by NSF Grants CCF-1049899 and CCF-1217770 to A. J. and M-Y. K.; and HIIT Pump Priming Grants No. 902184/T30606 and Academy of Finland, Postdoctoral Researcher Grants 13266670/T30606 to S. S.

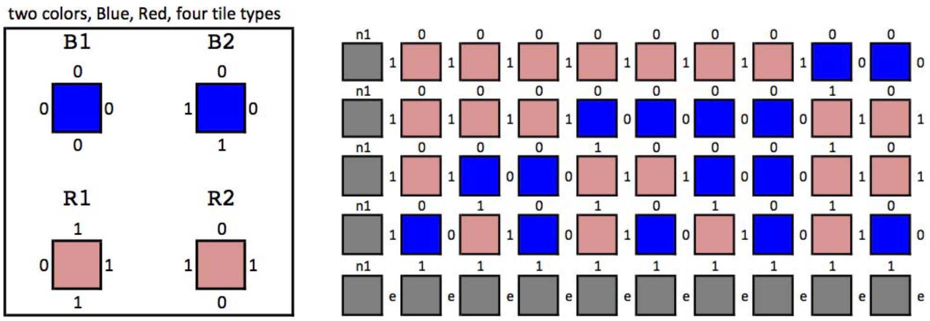


Fig. 1. We use 4 tile types, 2 colored Red and Blue each, to implement a half-subtractor. For each tile type with glues named (n,s,e,w), we see that its glues perform the operation "west minus south," then pass the output to the east and the carryout to the north. Interpreting Red types as 1 bits and Blue types as 0 bits, with MSBs at the top, we see that the seed's north glues in row 0 interact with this tile type set to successively subtract 1 from one column to the next.

Because of space constraints, most graphics and proofs needed for lower bounds and glue scheme enforcement appear only in the web version of the paper, which is also in full color.[1]

2 The Model and Preliminaries

We begin our sufficient description of Rectilinear Tile Assembly System (RTAS) with a proxy for DNA proteins and their "roles," and a method of "attachment." For a formal and excellent introduction, see [7].

A **(tile) type** is a unit-square with five parameters: four glues g_n, g_s, g_e, g_w associated with each edge north, south, east, and west; and a color c. A **tile** is an instance of a tile type. We assume tiles cannot be rotated or reflected. By inclusion of a particular tile type in a set, we assume that we have access to infinitely many tiles of the type. A **color** is an attribute of a type that can be shared by multiple types within a set- from a certain DNA perspective, it makes tile types indistinguishable and "serve the same role." A **glue** is an attribute of a type that is possibly equal to the glues of other types in a set, in which case the glues **match**- glues will tell us how we can "stick together" tiles.

A **(rectangular) assembly** (of width w and height h) is a function from a "rectangular" subset $\{1, ..., w\} \times \{1, ..., h\} \subset \mathbb{Z}_+^2$ to types such that for any pair of adjacent inputs that share an "edge," the output types' glues on the shared edge are the same. A **(rectangular) color pattern** (of width w and height h) is a function from a rectangle $\{1, ..., w\} \times \{1, ..., h\} \subset \mathbb{Z}_+^2$ to colors.

Our model for DNA self-assembly does not take place in a vacuum- we assume the preexistence of an L-shaped **seed**. We assume that our assembly function

[1] The full version of the paper is available at http://arxiv.org/

maps indexes in the 0-row and 0-column to seed types, which then have exposed north glues and east glues respectively. The seed tiles are colored Slate Gray in all of our graphics. The natural way to understand an assembly is to start with a seed, then the glues work to allow tiles to be attached as follows: A tile can **attach** at an index (i, j) if and only if its west glue matches the east glue of a tile already **placed** at $(i - 1, j)$ and its south glue matches the north glue of a tile placed at $(i, j - 1)$. Therefore an assembly builds up from south-west to north-east. If an assembly reaches a state such that no more tiles can attach, then it is a **terminal assembly**.

A **Rectilinear TAS** is a pair $\mathcal{T} = (T, \sigma_L)$ consisting of a tile type set T and an L-shaped seed σ_L. Its **size** is the cardinality of T. We will use and refer to the requirement that every two tile types must have distinct (south,west) glue-tuples as **uniqueness**. Two types that violate uniqueness are said to **clash**.

For RTASs satisfying uniqueness, we see by induction that fixing the exposed seed glues means that a tile placed at an empty index has necessarily deterministic type, and leads to a deterministic terminal assembly. As we care mostly for colors, we will also say it **uniquely assembles a color pattern**.

As illustrated in Fig. 1, we see that a blue tile `B2` uniquely attaches at index $(1, 1)$ given the exposed seed glues. The three tiles above it are successive copies of red tile type `R2` and, attaching in turn, none could be replaced by tiles of any other type. The assembly as shown is terminal.

The **Pattern self-assembly tile set synthesis (PATS)** problem was proposed by Ma and Lombardi [4,5]. It asks the question, "given a rectangular color pattern $\mathcal{P}$, what is a RTAS of minimum size that uniquely assembles the pattern?"

Definition 1. k-COLORED PATS (k-PATS)
INPUT: *a k-colored pattern $\mathcal{P}$;*
OUTPUT: *a smallest RTAS which uniquely self-assembles $\mathcal{P}$.*

60-PATS was shown to be $\mathcal{NP}$-hard by Seki [8]. We endeavor to show that 29-PATS is $\mathcal{NP}$-hard by a polynomial reduction from Subset Sum[2] to the size variant of 29-PATS: given a pattern in 29 colors, what is the minimum size of any RTAS that uniquely self-assembles the given pattern.
For the rest of this paper, we assume we have a black box capable of solving the size variant of 29-PATS.

3 The Circuit

We reduce an instance **SS** $= (S, n)$ of Subset Sum to a pattern in 29-colors `PATTERN` such that

1. There exists a RTAS (T, σ^*) with $|T| = 46$ that uniquely assembles `PATTERN` if and only if **SS** is solvable;

[2] We use the variant of Subset Sum that restricts all elements of a set S to be positive integers, and asks if any subset $S^* \subseteq S$ sums to some target $n \in \mathbb{N}$.

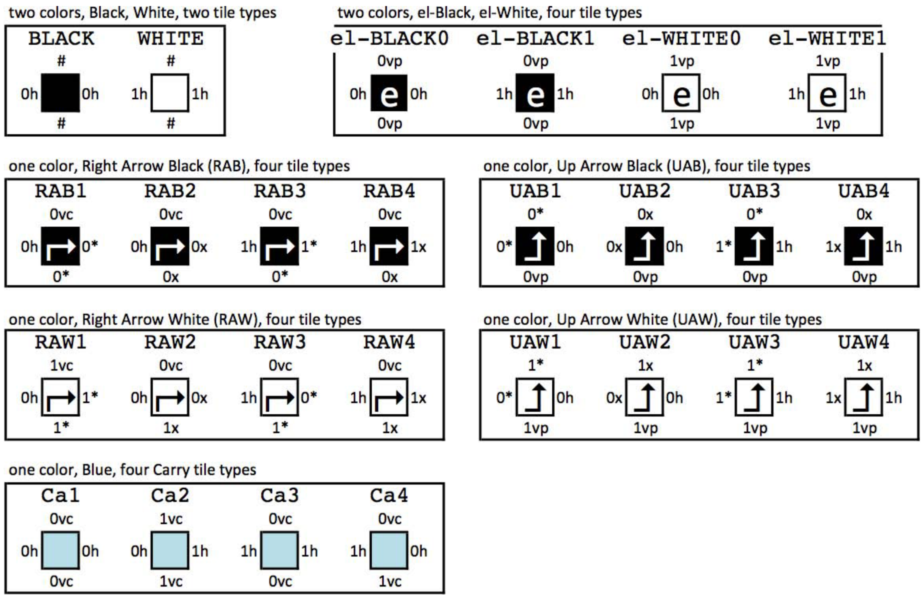

two colors, Black, White, two tile types
BLACK
WHITE
two colors, el-Black, el-White, four tile types
el-BLACK0
el-BLACK1
el-WHITE0
el-WHITE1
one color, Right Arrow Black (RAB), four tile types
RAB1
RAB2
RAB3
RAB4
one color, Up Arrow Black (UAB), four tile types
UAB1
UAB2
UAB3
UAB4
one color, Right Arrow White (RAW), four tile types
RAW1
RAW2
RAW3
RAW4
one color, Up Arrow White (UAW), four tile types
UAW1
UAW2
UAW3
UAW4
one color, Blue, four Carry tile types
Ca1
Ca2
Ca3
Ca4

Fig. 2. These are the 26 tiles types in 9 colors that currently make up the critical set T; they are sufficient to assemble `CIRCUIT` if an S^* solving **SS** exists.

2. There does not exist any tile type set $T' \neq T$ (ignoring isomorphism over glue names) of size 46 or less such that a RTAS (T', σ') can uniquely self-assemble `PATTERN`.

In this section, we verify (1) for a sub-pattern of `PATTERN` called `CIRCUIT` using 26 of the 46 tile types of T.[3] Specifically, we introduce a subset of the critical tile type set T in Fig. 2. An example of the reduction to `CIRCUIT` is given in Fig. 3. For solvable **SS**, T can uniquely assemble `CIRCUIT`, i.e., T is *sufficient*; therefore, $|T|$ is an upper bound on the output of the black box in these cases. The reduction works as follows:

We determine the height of the circuit to be $\max(\lceil \log_2(n + \Sigma_{\text{all}})\rceil, 21)$, where Σ_{all} is the sum of all elements in S.[4] Then $2^{\text{row}\#}$ is greater than both n and Σ_{all}. The width of `CIRCUIT` will be determined online. Then:

– The target number n from **SS** is encoded in binary- MSB at the top- as colors of tiles in the first column, Black for 0 and White for 1. Unique tile types of these colors in T imply that equivalently, n is also encoded in the east glues of this column.

[3] The other 20 tile types will be necessary and sufficient regardless of the instance **SS**.
[4] Later we add a sub-pattern which requires height of at least 21.

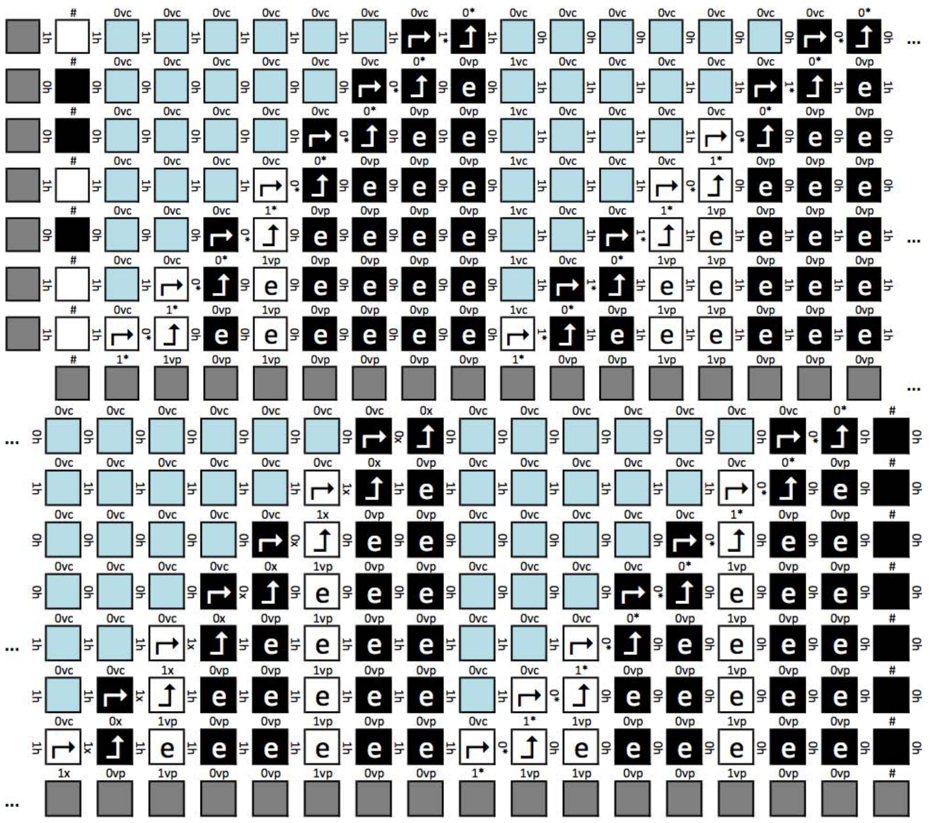


Fig. 3. For space considerations, and WLOG here, we draw this example with height of 7 rather than 21. **SS** = ($S = \{11, 25, 37, 39\}, n = 75$), the same example as Brun [3] for easy comparison of the tile type sets. $S^* = \{11, 25, 39\}$ so we see the seed north glue 1* in columns 2, 10 and 26; and the seed north glue 1x in column 18.

- Using (1 + row#) columns each, we similarly encode elements of S in succession, MSBs on the right. A 0 bit is encoded by a column of colors RAB, UAB, and el-Black; a 1 bit by RAW, UAW, and el-White.
- Given our reduction and tile type set T, there are only two colors appearing next to seed tiles in row 1 or column 1 such that the seed has a choice in glues: tile types of colors RAB and RAW have varying south glues. Compare this to the south glues of UAB for example, whose tile types all have south glue 0vp.
- Then given T, the set of feasible seeds can only vary in their choice of glues in the first column of the encoding of each element of S; the choice of seed is equivalently the choice to "tag" the glue '*' to mean we should subtract the current element, or 'x' to mean we should not.

- The RA and UA tile types cooperate to transfer the subtraction ON/OFF signal through an element's columns until it runs off the top.
- Subtraction of a bit takes place in the RA squares. Tile types below them preserve bit information of the element vertically. If subtraction is ON, the RA tile type works as a half-subtractor (like in Fig. 1): it receives a "running total" bit from the west, subtracts from it the south bit, outputs to the east and "carries" to the north. If subtraction is OFF, the RA tile type preserves bit information horizontally (passes it through) and carries 0.
- All indexes above the squares colored RAB and RAW are required to be color Carry Blue. These tile types are half-subtractors again, to complete the carry operation correctly.
- Below the RA colors, subtraction has already taken place so bit information is preserved horizontally for use by the next element of S to the right.
- The last column of `CIRCUIT` is all Black. We started with t, we subtracted some subset of elements of S, and for an instance of **SS** for which S^* exists, we should finish with all-Black 0.

Proposition 2. *Given an **SS** and our tile type set T, the RTAS (T, σ^*) with $|T| = 26$ uniquely assembles `CIRCUIT` if and only if **SS** is solvable.*

Proof. The reduction correctly encodes n and the elements of S, and correctly does subtraction or not. The proof is in the previous analysis, for further justification see Brun [3].

Then our choice over feasible seed glues is exactly equivalent to choices over subsets $S' \subseteq S$ of elements to subtract from n to try to obtain 0. □

4 Minimum Tile Complexity

In the last section, we described a tile type set T and showed that it is *sufficient* to self-assemble the color pattern `CIRCUIT` that results from reducing any satisfiable instance of Subset Sum, given an appropriate seed σ^*.

At this point, the size of T is an upper bound on what the black box can return on a satisfiable input `CIRCUIT`. But in a trivial example in which the elements of S sum to n exactly, we have no use for any tile types utilizing the glues 0x or 1x so output will be less. Worse, it is possible that given an unsolvable instance, the black box outputs 26 or less using some other T', making it indistinguishable from a solvable instance.

So we have a need to establish a lower bound on outputs from the black box. To do this, we add the tile types in Fig. 4 to T.

We will join `CIRCUIT` with auxiliary patterns into the full input `PATTERN`; then for each color c that appears in `PATTERN`, we will prove that the number of tile types colored c needed by any set T' that uniquely assembles `PATTERN` must be at least the number used by T of color c. Finally, summing over types will give us our lower bound- the black box cannot output less than 46.

Auxiliary patterns must meet the following constraints: they must assemble using T, i.e., be **consistent**; they must also "splice" together using T. Failing

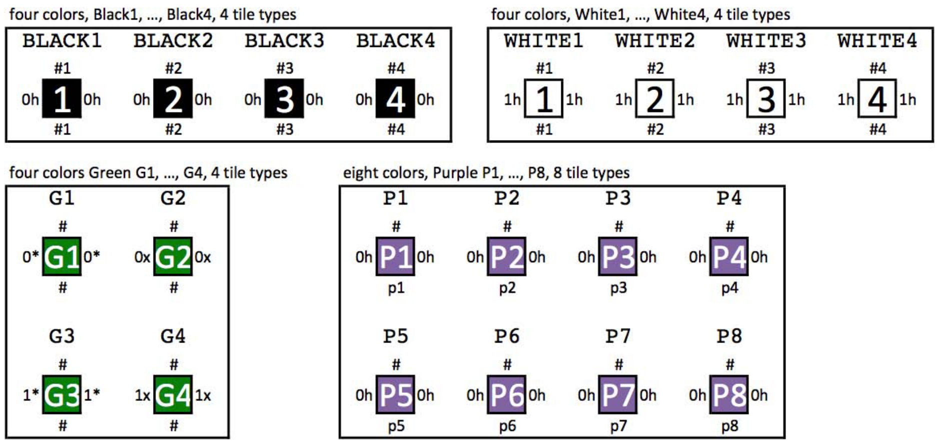

four colors, Black1, ..., Black4, 4 tile types
BLACK1 BLACK2 BLACK3 BLACK4
#1 #2 #3 #4
0h 1 0h
0h 2 0h
0h 3 0h
0h 4 0h
four colors, White1, ..., White4, 4 tile types
WHITE1 WHITE2 WHITE3 WHITE4
1h 1 1h
1h 2 1h
1h 3 1h
1h 4 1h
four colors Green G1, ..., G4, 4 tile types
G1 G2 G3 G4
0* G1 0*
0x G2 0x
1* G3 1*
1x G4 1x
#
eight colors, Purple P1, ..., P8, 8 tile types
P1 P2 P3 P4 P5 P6 P7 P8
0h P1 0h
p1 p2 p3 p4 p5 p6 p7 p8

Fig. 4. We add these 20 tile types to our set T. While they are not necessary to uniquely assemble `CIRCUIT`, adding them does not affect the sufficiency of T, now of size 46; they will be used in auxiliary patterns that will be attached to `CIRCUIT` such that all 46 tiles are necessary.

either, we lose the sufficiency of T. However, these constraints endow a benefit: because T is sufficient to assemble `CIRCUIT` and all auxiliary patterns for reductions from satisfiable instances of Subset Sum, we know for these cases that the black box need not consider tile type sets larger than 46.

So the goal of this section and the next is to design auxiliary sub-patterns such that, including them in `PATTERN`, the tile set T will be *necessary.* Sub-patterns in this section are named `LB#` for Lower Bound. We claim that T is sufficient for all auxiliary patterns in Sections 4 and 5 without proof. The strategy for joining together the various sub-patterns is given as part of correctness in Section 6.

We take it as a given that the 29 colors described in T will be used in some sub-pattern of `PATTERN`. There are 22 colors in `PATTERN` with just one tile type of their color in T.[5] Each trivially requires one type, so we have finished 22 of our 29 necessary lower bound proofs, leaving 17 tiles unassigned, or **free**.

Sub-pattern `LB1` is given in Fig. 5 and is used in the next result. For space considerations, we provide this one result and jump to conclude Section 4.

Proposition 3. *If `LB1` appears as a sub-pattern of `PATTERN`, then in any minimum tile type set solving `PATTERN`, either there must be at least 2 tile types with distinct east glues of each color el-Black, el-White, and Carry Blue as we suggest in T; or the black box must find a minimum tile type set larger than 46.*

Proof. By symmetry, we only need to prove the case for el-Black. We assume that the black box can find a solution using at most 46 tile types.

[5] Colors Black and White from Fig. 2, and all 20 colors from Fig. 4.

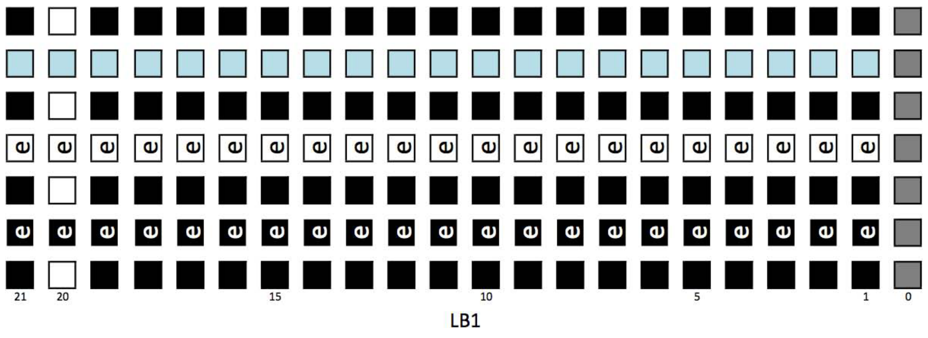


Fig. 5. `LB1` is presented sideways for space considerations. The Slate seed tiles should be interpreted as being the 0-row, then it is the rows that are numbered up to its height of 21. `LB1` guarantees that there are at least 2 tile types colored each of el-Black, el-White, and Carry Blue.

By contradiction, assume there is only one tile type el-Black. Then there is only one glue east of column 2, call it e. The Black types can have at most 18 unique north glues in column 3, so by the 19th north glue, they must cycle (because the west glue is a constant). But this requires a Black tile at $(3, 20)$, not White. □

Proposition 4. *If `LB1` through `LB118` appear as sub-patterns of `PATTERN`, then for any minimum tile type set solving `PATTERN`, it must have total size at least 46, with the desired lower bounds holding for each color, as we suggest in T.*

Proof. See the full version of the paper for intermediate results and final proof.

5 Glue Interpretations

In this section, we show that if T has size 46, then the glues of some tile types in T must be isomorphic to the names we give to them in Fig. 2. It is sufficient for our purposes to only prove the glue scheme for tile types used in `CIRCUIT`[6] because then it is clear that for unsolvable **SS**, T cannot assemble `CIRCUIT` in 26 tile types. For convenience, and WLOG, we use the type-naming and glue-naming schemes in T. New patterns are named `GE#`, for Glue Enforcement.

We present `LB42` in Fig. 6 for the next two propositions. For space considerations, we present these two results and jump to conclude Section 5.

Proposition 5. *If `LB1` through `LB118` appear as sub-patterns of `PATTERN` (and establish a minimum tile type set size of 46), then in any minimum tile type set solving `PATTERN`, either the glues of the Black and White tile types are as we suggest in T; or the black box must find a minimum tile set larger than 46.*

[6] Though we do prove glues for the Green tile types to facilitate later results.

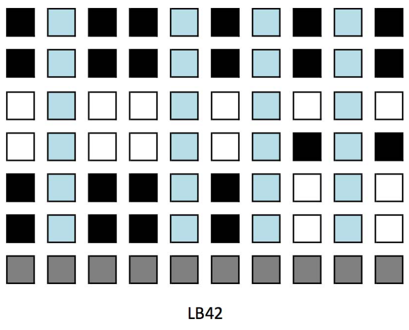


Fig. 6. Blue Carry tiles act as half-subtractors, an entire column of them can subtract 0 or 1. It is straightforward to force the glues for Black, White, and Carry tile types.

Proof. We assume that the black box can find a solution using at most 46 tile types. Then there must be exactly 1 type of each color Black and White, as we exhaust 46 tile types to satisfy all lower bounds established in Section 4.

Now we see in `LB42` that `BLACK` self-stacks both vertically and horizontally, therefore we know that it has the same north and south glues, call it #, and the same east and west glues, call it 0h.[7] `WHITE` stacks both above and below `BLACK`, so its vertical glues are the same.

`WHITE` also self-stacks horizontally, so it also has its east glue equal to its west glue, call it 1h. However by uniqueness, the glues 1h and 0h must be distinct, because of the common # south glues of the tile types. □

Proposition 6. *If `LB1` through `LB118` appear as sub-patterns of `PATTERN`, then in any minimum tile type set solving `PATTERN`, either the glues of the Carry Blue tile types are as we suggest in T; or the black box must find a minimum tile set larger than 46.*

Proof. We assume that the black box can find a solution using at most 46 tile types. Then there must be exactly 4 tile types of color Blue, as we exhaust 46 tile types to satisfy all lower bounds established in the last section.

Now using Proposition 5 we see in `LB42` that four Blue tile types necessarily have (east,west) glue tuples known to be distinct. For example, tiles placed at indexes $(2,2), (7,2), (2,3)$, and $(7,3)$ require their glues to be (0h,0h), (0h,1h), (1h,1h), and (1h,0h) respectively. This nicely exhausts the Blue types available. Name the distinct types that end up at these indexes `CA1` through `CA4` in order.

We notice that `CA1` stacks with itself at $(2,1)$ and $(2,2)$ (because we have shown that Carry tile types are identifiable by their (west,east) glue tuples), and `CA3` stacks with itself at $(2,3)$ and $(2,4)$, and they stack both above and below each other. Therefore the north and south glues of both types must all be

[7] We use "h" for horizontal; and soon "v" for vertical.

equal, call it 0vc.[8] Note that this glue 0vc must also be present between $(7, 3)$ and $(7, 4)$, so it is also the north glue of `CA4`.

Now we look at `CA2` and notice that it stacks with itself at $(7, 1)$ and $(7, 2)$. Therefore its north and south glues must be equal; and additionally they must be the same as the south glue of `CA4`. However, they cannot be 0vc, otherwise the `CA2` tile attaching at $(7, 2)$ would not do so uniquely (it would clash with `CA1` at $(3, 2)$). So this second group of glue assignments must be distinct, call it 1vc. Neither 0vc nor 1vc can be the same as #, by uniqueness. □

Proposition 7. *If `GE1` through `GE12` appear as sub-patterns of `PATTERN`, along with `LB1` through `LB118`, then in any minimum tile type set solving `PATTERN`, the glues of all tile types that can be used in `CIRCUIT` must be the same as we suggest in T, up to isomorphic symmetry; or the black box must find a minimum tile type set larger than 46.*

Proof. See the full version of the paper for intermediate results and final proof.

6 Integration and Correctness

In this Section we prove that 29-PATS is $\mathcal{NP}$-hard. We start by addressing a minor concern in the design- the possibility that the black box might choose a set of elements to subtract such that the running total "becomes negative" after subtracting some element; then it might keep subtracting elements and reach 0, though this subset does not actually sum to the original target n.

Our design precludes this possibility by requiring the height of the pattern, equal to $\max(\lceil \log_2(n + \Sigma_{\text{all}}) \rceil, 21)$, to be sufficiently large: if the running total ever "goes negative," then even by subtracting every remaining element of S, we cannot possibly make it back down to 0 again.

When an element is subtracted from the running total, the operation that we are actually performing is subtraction-mod-$2^{\text{row}\#}$, and we use this idea to join together sub-patterns. We can verify that every color sub-pattern begins and ends with a column of squares colored by Black and White, or their copies. Then *every east glue and every west glue of every sub-pattern is either 0h or 1h.* Taking the italicized text as an invariant gives us the needed result.

Proposition 8. *Given 2 sub-patterns P_1 and P_2 of the same height row# such that all of the east glues of P_1 and all of the west glues of P_2 are 0h or 1h, we can join them together into one sub-pattern P^* using the tile types of T.*

Proof. As just explained, our sub-pattern that represents the subtraction of one element from a number "written" vertically is actually subtraction-mod-$2^{\text{row}\#}$. Then we are free to think of both the east glues of P_1 and the west glues of P_2 as respectively input and output numbers mod $2^{\text{row}\#}$, and we can design the needed subtraction-mod-$2^{\text{row}\#}$ to take place in between them. □

[8] Along with "v" for vertical, we use "c" for carry; and soon "p" for passthrough.

We finally have a full description of `PATTERN`. It is the result of listing `CIRCUIT`, `LB1` through `LB118`, and `GE1` through `GE12` in order and then joining successively listed sub-patterns together as described in Proposition 8.

Theorem 9. *For any instance of Subset Sum* ***SS****, if we reduce it to the 29-color pattern* `PATTERN` *and input it to our black box, the size of a minimum tile type set that we receive as output will be equal to 46 if and only if* ***SS*** *is satisfiable.*

Proof. From Proposition 4 we know that the minimum tile set must have size at least 46; and from Proposition 7 we know that if a tile set has size 46 and assembles the reduction, then it must be isomorphic to T, i.e., T is necessary.

From Proposition 2, we know that T is sufficient for any solvable **SS**. Necessity and sufficiency of T for solvable **SS** completes the "if" direction of our result. Now we show that T is not sufficient if **SS** is not solvable.

We start by analyzing the extent of the ability of the black box to actually make decisions about seed glue choice for the sub-pattern `CIRCUIT`, if it intends to use only 46 tiles. From our discussion in Section 3, we know that it is only for the southern glues of RAB and RAW tiles in the first row that the black box actually gets to make a choice. For RAB tiles it can choose 0* or 0x; for RAW it can choose 1* or 1x. Again by Proposition 2 these choices correctly and exactly correspond to subtracting an element or not.

By the assumption that we are working with an instance **SS** that is unsolvable, there is no sequence of seed glue choices for RAB and RAW tiles such that the east glues of the second-to-last column in `CIRCUIT` are all 0h, to match the uniformly 0h west glues required by `CIRCUIT`'s last column's Black squares.

But T is forced when we are restricted to at most 46 tile types; it must be the case that for inputs `PATTERN` that result from reducing unsolvable instances of **SS**, the black box finds a minimum tile set of size strictly larger than 46. □

The statement of Theorem 9 combines the goals given at the beginning of Section 3. It directly implies Theorem 1 because an instance of Subset Sum **SS** is solvable if and only if the black box outputs a minimum RTAS size of 46 for an input of `PATTERN`, completing the reduction.

Corollary 10. *The original PATS problem cannot be approximated to within a 47/46 ratio.*

Corollary 11. *k-PATS is not in PTAS for any* $k \geq 29$.

References

1. Adleman, L.: Towards a mathematical theory of self-assembly. Tech. Rep. 00-722, USC (2000)
2. Barish, R., Schulman, R., Rothemund, P.W.K., Winfree, E.: An Information-bearing seed for nucleating algorithmic self-assembly. Proc. Natl. Acad. Sci. 106, 6054–6059 (2009)
3. Brun, Y.: Solving $\mathcal{NP}$-Complete Problems in the Tile Assembly Model. Theo. Comp. Sci. 395, 31–46 (2008)

4. Ma, X., Lombardi, F.: Synthesis of tile sets for DNA self-assembly. IEEE T. Comput. Aid. D. 27(5), 963–967 (2008)
5. Ma, X., Lombardi, F.: On the computation complexity of tile set synthesis for DNA self-assembly. IEEE T. Circuits-II 56(1), 31–35 (2009)
6. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic Self-Assembly of DNA Sierpinski Triangles. PLos Biology 2, 2041–2053 (2004)
7. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Proc. of STOC 2000, pp. 459–468 (2000)
8. Seki, S.: Combinatorial Optimization in Pattern Assembly. In: Mauri, G., Dennunzio, A., Manzoni, L., Porreca, A.E. (eds.) UCNC 2013. LNCS, vol. 7956, pp. 220–231. Springer, Heidelberg (2013)
9. Winfree, E.: On the Computational Power of DNA Annealing and Ligation. DNA Based Computers, 199–221 (1996)
10. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (June 1998)
11. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 539–544 (1998)

A Probabilistic Analysis of Kademlia Networks*

Xing Shi Cai and Luc Devroye

School of Computer Science, McGill University of Montreal, Canada,
xingshi.cai@mail.mcgill.ca,
lucdevroye@gmail.com

Abstract. Kademlia [3] is currently the most widely used searching algorithm in P2P (peer-to-peer) networks. This work studies an essential question about Kademlia from a mathematical perspective: how long does it take to locate a node in the network? To answer it, we introduce a random graph $\mathcal{K}$ and study how many steps are needed to locate a given vertex in $\mathcal{K}$ using Kademlia's algorithm, which we call the *routing time*. Two slightly different versions of $\mathcal{K}$ are studied. In the first one, vertices of $\mathcal{K}$ are labeled with fixed IDs. In the second one, vertices are assumed to have randomly selected IDs. In both cases, we show that the routing time is about $c \log n$, where n is the number of nodes in the network and c is an explicitly described constant.

1 Introduction to Kademlia

A P2P (peer-to-peer) network [4] is a computer network which allows sharing of resources like storage, bandwidth and computing power. Unlike traditional client-server architectures, in P2P networks, a participating computer (a *node*) is not only a consumer but also a supplier of resources. Nowadays, major P2P services in the internet often have millions of users. For an overview of P2P networks, see Steinmetz [5].

The huge size of P2P networks raises one challenge—among millions of nodes, how can a node find another one efficiently? To address this, a group of algorithms called DHT (Distributed Hash Table) [6] was invented in the early 2000s, including Pastry [7], CAN [8], Chord [9], Tapestry [10], and Kademlia [3]. Created by Maymounkov and Mazières in 2002, Kademlia has become the de facto standard searching algorithm for P2P services. It is used by BitTorrent [11] and the Kad network [12], both of which have more than a million nodes.

Kademlia assigns each node an ID chosen uniformly at random from $\{0,1\}^d$, the d-dimension hypercube, where d is usually 128 [12] or 160 [11]. Thus we *always* refer to a node by its ID. Given two IDs $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$, Kademlia defines their XOR *distance* by

$$\delta(x,y) = \sum_{i=1}^{d} (x_i \oplus y_i) \times 2^{d-i},$$

* We would like to thank Carlton Davis, José M. Fernandez, (authors of [1] and [2]) and Mahshid Yassaei for collaborations and explanations regarding Kademlia.

where $\oplus$ denotes the XOR operation

$$u \oplus v = \begin{cases} 1 & \text{if } u \neq v, \\ 0 & \text{otherwise.} \end{cases}$$

Note that distance and closeness *always* mean XOR distances between IDs in this work.

Since it is almost impossible for a node to know where all other nodes are located, in Kademlia a node, say x, is only responsible for maintaining a table (a *routing table*) for a small number of other nodes (x's *neighbors*). Roughly speaking, $\{0, 1\}^d$ is partitioned into d subsets, such that nodes in the same subset have similar distances to x. Within each subset, up to k nodes' information is recorded in a list (a k-*bucket*), where k is a constant which usually equals 8 [11], 10 [12] or 20 [13]. All of x's k-buckets form x's routing table.

When x needs to locate node y which is not in its routing table, x sends queries to α of its neighbors which are closest to y, where α is a constant, sometimes chosen as 3 [14] or 10 [13]. A recipient of x's message returns locations of k of its own neighbors with shortest distance to y. With this information, x again contacts α nodes that are closest to y. This approach (*routing*) repeats until no one closer to y can be found. The efficiency of routing is critical for the overall performance of Kademlia. Its analysis is the topic of this work.

2 Our Model

Consider a Kademlia network of n nodes $X_1, \ldots, X_n$. Writing IDs as strings consisting of zeros and ones, from higher bits to lower bits, we can completely represent $X_1, \ldots, X_n$ in a binary *trie*, as depicted in Fig. 1. A trie is an ordered tree data structure invented by Fredkin [15]. For more on tries, see Szpankowski [16]. Paths are associated with bit strings—0 corresponds to a left child, and 1 to a right child. The bits encountered on a path of length d to a leaf is the ID, or value, of the leaf. In this manner, the binary trie, has height d, and precisely n leaves of distance d from the root.

Let $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ be two IDs (leaves) in the trie. Let $\ell(x, y)$ be the length of the path from the root to x and y's lowest common ancestor, i.e., the length of x and y's common prefix. We have

$$\ell(x, y) = \max\{i : x_1 = y_1, \ldots, x_i = y_i\} \ .$$

It is easy to verify that $\ell(x, y)$ bounds the distance of x and y by

$$2^{d-\ell(x,y)-1} \leq \delta(x, y) < 2^{d-\ell(x,y)} \ .$$

Thus if we partition $\{0, 1\}^d \setminus \{x\}$ according to distances to x as follows

$$\mathcal{D}_i(x) = \{y : 2^{i-1} \leq \delta(x, y) < 2^i\}, \quad i = 1, \ldots, d,$$

then $\mathcal{D}_i(x)$ is equivalent to a subtree in the trie, in which each node shares a common prefix of length i with x. Therefore, we have the equivalent definition

$$\mathcal{D}_i(x) = \{y : \ell(x, y) = d - i\}, \quad i = 1, \ldots, d \ .$$

Let $\mathcal{B}_i(x)$ be the set of IDs in the k-bucket of x corresponding to $\mathcal{D}_i(x)$. In our model, we assume that if $|\mathcal{D}_i(x)| \le k$, then $\mathcal{B}_i(x) = \mathcal{D}_i(x)$. Otherwise, for all $A \subset \mathcal{D}_i(x)$ with $|A| = k$, we have

$$\mathbb{P}\left\{\mathcal{B}_i(x) = A\right\} = \binom{|\mathcal{D}_i(x)|}{k}^{-1} \ .$$

In other words, we fill up each k-bucket uniformly at random without replacement.

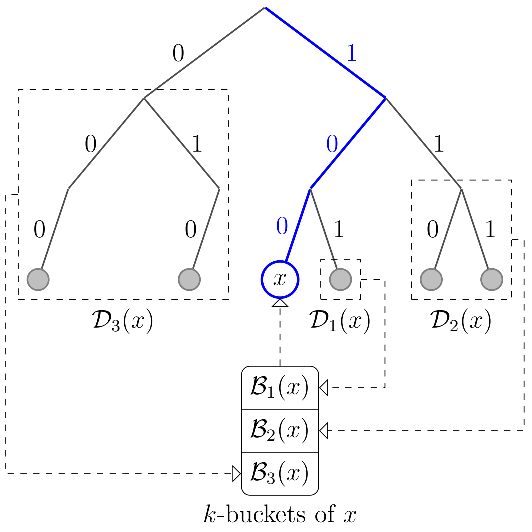


Fig. 1. An example of Kademlia ID trie and k-buckets. Given an ID $x = (1, 0, 0)$, the trie is partitioned into subtrees $\mathcal{D}_1(x), \mathcal{D}_2(x), \mathcal{D}_3(x)$. Node x maintains a k-bucket for each of these subtrees containing the information of up to k nodes in the subtree, which we denote by $\mathcal{B}_1(x), \mathcal{B}_2(x), \mathcal{B}_3(x)$ respectively.

Consider a directed graph $\mathcal{K}$ with vertex set $\mathcal{V} = \{X_1, \ldots, X_n\}$. Let its edge set be

$$\mathcal{E} = \{(u, v) : u, v \in \mathcal{V},\ v \in \cup_{i=1}^{d} \mathcal{B}_i(u)\} \ .$$

Put differently, we add a directed edge (u, v) in $\mathcal{K}$ if and only if v is in one of u's k-buckets. When $\alpha = 1$, only one node is queried at each step of routing, the search process starting at x for y can be seen as a path ρ_{xy} in $\mathcal{K}$ (the *routing path*). It starts from vertex x, then jumps to the vertex that is closest to y among all x's neighbors. From there, it again jumps to the adjacent vertex that is closest to y. Let y^* be the unique vertex with the shortest distance to y in $\mathcal{K}$. Since the distance between the current vertex and y^* decreases at each step until zero, ρ_{xy} has no loop and always ends at y^*. In fact, $\mathcal{K}$ is always strongly connected.

Let T_{xy} be the path length of ρ_{xy}. Since T_{xy} equals the number of rounds of messages x needs to send before routing ends, we call it the *routing time*. Our first main result assumes that $X_1 = x_1, \ldots, X_n = x_n$, where $x_1, \ldots, x_n$ are all fixed d-bit vectors, which we call the *deterministic* ID *model*. The randomness thus comes only from the filling of the k-buckets. We always assume that $d \geq \log_2 n$, and we let $n \to \infty$ (and thus $d \to \infty$) in our asymptotic results. (Note that in this work $\log n$ denotes the natural logarithm of n.)

Theorem 1. *In the deterministic* ID *model, we have*

$$\sup_{x_1,\ldots,x_n} \sup_x \sup_y \mathbb{E}\left[T_{xy}\right] \leq (c_k + o(1)) \log n,$$

$$\sup_{x_1,\ldots,x_n} \sup_x \mathbb{E}\left[\sup_y T_{xy}\right] \leq (c'_k + o(1)) \log n,$$

$$\sup_{x_1,\ldots,x_n} \mathbb{E}\left[\sup_x \sup_y T_{xy}\right] \leq (c^*_k + o(1)) \log n,$$

where c_k, c'_k, c^*_k are constants depending only on k. In particular, we have $c_k = 1/H_k$, where $H_k = \sum_{i=1}^{k} 1/i$, also known as the k-th harmonic number.

The first inequality in this theorem gives an upper bound over the expected routing time between two fixed nodes. Since $c_k \leq c_1 \leq 1/\log 2$, the first bound is better than the $\lceil \log_2 n \rceil$ bound described by Maymounkov and Mazières in the original Kademlia paper [3]. The second inequality considers the expectation of the maximal routing time when the starting node is fixed, and a look-up is performed for each of the n destinations. The third one considers the expectation of the maximal routing time in the whole network if all n nodes were to look up all n destinations.

Our second main result considers the situation when $X_1, \ldots, X_n$ are selected uniformly at random from $\{0,1\}^d$ without replacement (the *random* ID *model*). Given an ID x, let $\widehat{x}$ denote the ID that is farthest away from x (x's *polar opposite*). Since by symmetry $T_{X_1\widehat{X}_1}, T_{X_2\widehat{X}_2}, \ldots T_{X_n\widehat{X}_n}$ are identically distributed, we only need to study $T_{X_1\widehat{X}_1}$, which we denote by $\widehat{T}$.

Theorem 2 ([17]). *In the random* ID *model, we have*

$$\frac{\widehat{T}}{\log n} \to \frac{1}{g(k)} \qquad \textit{in probability},$$

as $n \to \infty$, where $g(k) = H_k + O(1)$ is a function of k.

Since $\widehat{T} = T_{X_1\widehat{X}_1}$, we see that

$$\frac{1}{g(k)} \leq \frac{1}{H_k} \ .$$

However, we have almost identical behavior because $g(k) = H_k + O(1)$ as $k \to \infty$. Let $d = d(n) \geq \log_2 n$. By the probabilistic method, Theorem 2 implies that for every $\epsilon > 0$, there exists for every n a non-repetitive sequence $(x_1(n), \ldots, x_n(n)) \in (\{0,1\}^d)^n$ of deterministic IDs such that with probability going to one,

$$T_{X_1 \widehat{X}_1} \geq \left(\frac{1}{g(k)} - \epsilon \right) \log n \ .$$

In view of $g(k) = H_k + O(1)$, we thus see that the first bound of Theorem 1 is almost tight.

Recall that y^* denotes the node that is closest to ID y. If we look at the lowest common ancestor of y^* and each hop of ρ_{xy}, then searching y from x can be seen as travelling downwards along the path from the root to the leaf y^* in the trie, with the distance of each hop being random. In the random ID model, when n is large, we can approximate this process in a full binary trie with infinite depth. In Sect. 4, we sketch the proof of Theorem 2 which uses this method. This does not work in the deterministic ID model as the structure of the ID trie can be unbalanced. But in Sect. 3, we show that there is another way to bound the routing time.

Due to its success, Kademlia has aroused great interest among researchers. But this is the first time that it is studied from a mathematical perspective. Our results point out one important reason for the success of Kademlia — its routing algorithm, while being extremely simple, works surprisingly well. This work also shows that probabilistic methods together with the right choice of a data structure, a trie in our case, could significantly simplify the analysis of a problem which was previously considered too troublesome to analyze rigorously.

3 The Deterministic ID Model

In this section, we assume that $X_1 = x_1, \ldots, X_n = x_n$, where $x_1, \ldots, x_n$ are fixed d-bits vectors. Note that the distribution of T_{xy} is decided only by distances between vertices and the distance between x and y. Thus, by rotating the hypercube, we can always assume y to be a specific ID, which we choose $\bar{1} = (1, 1, \ldots, 1)$.

Figure 2 depicts the first hop of $\rho_{x\bar{1}}$ as jumping from one leaf to another in the ID trie. It is easy to see that if we always arrange branches representing 1 to the right hand side, which we take as a convention, then the closer a leaf is to the right, the closer it is to $\bar{1}$. Thus the rightmost leaf in the trie, which we always denote by y', is closest to $\bar{1}$ and is thus the end point of $\rho_{x\bar{1}}$.

Write $\rho_{x\bar{1}} = (z_0, z_1, \ldots)$ where $z_0 = x$. Let $i = d - \ell(z_0, \bar{1})$. We can see from Fig. 2 that z_1, the first hop, must belong to $\mathcal{D}_i(z_0)$, the highest subtree on the right hand side of z_0, which we denote by S_0. Since being z_0's neighbor implies membership in one of z_0's k-buckets, we have $z_1 \in \mathcal{B}_i(z_0) \subseteq S_0$. Recalling how $\mathcal{B}_i(z_0)$ is decided, we can think of the first hop as selecting up to k leaves from S_0 uniformly at random and choosing the rightmost one as z_1. Thus we can define of $\rho_{x\bar{1}}$ recursively as follows:

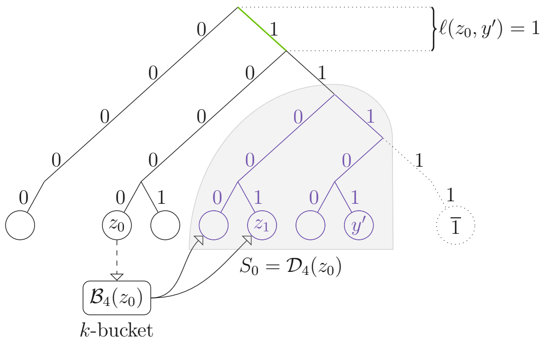


Fig. 2. An example of the first hop of $\rho_{x\bar{1}}$ with $d = 5$, $k = 2$. Since $d - \ell(z_0, y') = 4$, z_1 must be in subtree $S_0 = \mathcal{D}_4(z_0)$, which is the highest subtree to the right of z_0. We choose up to k leaves from S_0 uniformly at random without replacement, and let z_1 be the rightmost one.

– Let $z_0 = x$. Repeat the following step.
– Given z_t and $t \geq 0$, let S_t be highest subtree on the right hand side of z_t. If $S_t = \emptyset$, terminate. Otherwise select up to k leaves from S_t uniformly at random without replacement, and let the rightmost one be z_{t+1}.

Since $S_0 \supset S_1 \supset \cdots \supset S_{T_{x\bar{1}}} = \emptyset$, by studying how quickly the sequence $(|S_t|)_{t\geq 0}$ decreases to 0, we can bound how big $T_{x\bar{1}}$ could be. Although it is difficult to write the distribution of $(|S_t|)_{t\geq 0}$, we can approximate it with another sequence $(W_t)_{t\geq 0}$. Let $B(k)$ be the minimum of k independent uniform $[0, 1]$ random variables. Let $(B_t)_{t\geq 0}$ be a sequence of i.i.d. random variables with distribution $B(k)$. We define $W_t = |S_0| \times \prod_{s=1}^{t} B_s$.

Given two random variables A and B, we say A is *stochastically smaller than* B, denoted by $A \preceq B$, if and only if

$$\mathbb{P}\{A \geq r\} \leq \mathbb{P}\{B \geq r\} \qquad \text{for all } r \in \mathbb{R},$$

where $\mathbb{R}$ is the set of real numbers. The random variable W_t is stochastically larger than $|S_t|$, as there is a "trimming" effect at each hop. For example, the number of leaves between z_1 and y' has a distribution similar to $\lfloor W_1 \rfloor$. But some of these leaves might not belong to S_1.

Lemma 1. *For all $t \geq 1$, we have $|S_t| \preceq W_t$.*

Lemma 2. *For all $t \in \mathbb{N}$, we have:*

$$\text{(i)} \qquad \sup_{x_1,\ldots,x_n} \sup_{x} \sup_{y} \mathbb{P}\{T_{xy} \geq t\} \leq \mathbb{P}\{nB_1 \ldots B_t \geq 1\},$$

$$\text{(ii)} \qquad \sup_{x_1,\ldots,x_n} \sup_{x} \mathbb{P}\left\{\sup_{y} T_{xy} \geq t\right\} \leq n \times \mathbb{P}\{nB_1 \ldots B_t \geq 1\},$$

$$\text{(iii)} \qquad \sup_{x_1,\ldots,x_n} \mathbb{P}\left\{\sup_{x} \sup_{y} T_{xy} \geq t\right\} \leq n^2 \times \mathbb{P}\{nB_1 \ldots B_t \geq 1\} \quad .$$

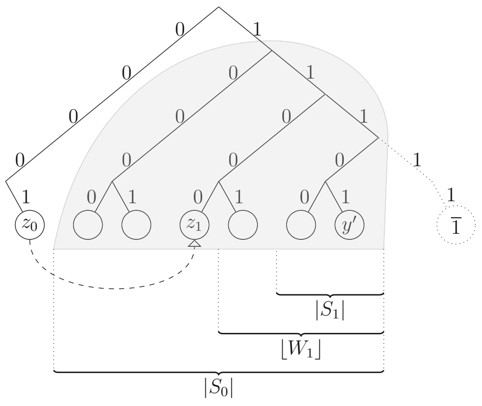


Fig. 3. The “trimming” effect

A beta random variable $B(a, b)$ has probability density function

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\, x^{a-1}(1-x)^{b-1}, \quad 0 \le x \le 1,$$

where $\Gamma(z)$ is the gamma function $\Gamma(z) = \int_0^\infty e^{-t}t^{z-1}\,\mathrm{d}t$. In order statistics theory, a basic result [18, chap.2.3] is that the r-th smallest of m i.i.d. uniform random variables has beta distribution $B(r, m+1-r)$. Plugging in $r = 1, m = k$, we have $B_t \overset{\mathcal{L}}{=} B(k) \overset{\mathcal{L}}{=} B(1, k)$ for all $t \in \mathbb{N}$. (For more about beta distribution, see [19, chap.25].) It is easy to check that for all $r > 0$ and $t \in \mathbb{N}$, we have

$$\mathbb{E}\left[(B_1 \cdots B_t)^r\right] = \mathbb{E}\left[B_1^r\right]^t = \mathbb{E}\left[B(1,k)^r\right]^t = \left(\frac{k!}{\prod_{i=1}^k (r+i)}\right)^t . \qquad (1)$$

By applying this moment bound, we have the following theorem:

Theorem 3. *There exist constants c_k, c'_k and c^*_k such that:*
(i) for all $c > c_k$,

$$\sup_{x_1,\ldots,x_n} \sup_x \sup_y \mathbb{P}\left\{T_{xy} \ge c\log n\right\} \to 0 \qquad \text{as } n \to \infty;$$

(ii) for all $c > c'_k$,

$$\sup_{x_1,\ldots,x_n} \sup_x \mathbb{P}\left\{\sup_y T_{xy} \ge c\log n\right\} \to 0 \qquad \text{as } n \to \infty;$$

(iii) for all $c > c^*_k$,

$$\sup_{x_1,\ldots,x_n} \mathbb{P}\left\{\sup_{x,y} T_{xy} \ge c\log n\right\} \to 0 \qquad \text{as } n \to \infty .$$

In particular, we have $c_k = 1/H_k$ where H_k is the k-th harmonic number.

It is easy to check that $c'_1 = e$, since $(r+1)/\log(1+r)$ takes minimum value e when $r = e-1$. But unlike c_k, we do not have closed forms for c'_k and c^*_k. Table 1 shows the numerical values of c_k, c'_k, c^*_k for $k = 1, \ldots, 10$.

Table 1. The numerical values of c_k, c'_k, c^*_k for $k = 1, \ldots, 10$

k	c_k	c'_k	c^*_k
1	1	2.718281828	3.591121477
2	0.6666666667	1.673805050	2.170961287
3	0.5454545455	1.302556173	1.668389781
4	0.4800000000	1.105969343	1.403318015
5	0.4379562044	0.9817977138	1.236481558
6	0.4081632653	0.8950813294	1.120340102
7	0.3856749311	0.8304602569	1.034040176
8	0.3679369251	0.7800681679	0.9669189101
9	0.3534857624	0.7394331755	0.9129238915
10	0.3414171521	0.7058123636	0.8683482160

Lemma 3. *We have*

$$\lim_{k\to\infty} c_k \log k = \lim_{k\to\infty} c'_k \log k = \lim_{k\to\infty} c^*_k \log k = 1 \ .$$

Remark 1. We are not providing precise inequalities with matching lower bounds. This can be done, but in that case, one could have to distinguish between many choices for d. We have already noted that $d \geq \log_2 n$. We always have

$$T_{xy} \leq d \ .$$

Therefore, for d very large, there is a danger of having routing times that are super-logarithmic in n. Our analysis shows that this is not the case. However, the precise behavior of T_{xy}, uniformly over all x,y and d, requires additional analysis. The behavior for d near $\log_2 n$, $d = \Theta(\log n)$, and $d/\log n \to \infty$ is quite different.

Remark 2. The performance bounds of this section are of the form $c_k \log n$ with $c_k = 1/H_k$. Although formulated for fixed k, they remain valid if k is allowed to depend upon n. For example, if $k = \log n$—that is, the routing table size grows as $d \times \log n$—the expected routing time is bounded by

$$(1+o(1))\frac{\log n}{\log k} \sim (1+o(1))\frac{\log n}{\log\log n} \ .$$

Even more important is the possibility of having $O(1)$ routing time. With $k \sim n^\theta$ for $\theta \in (0,1)$, the routing table size for one computer grows as $d \times n^\theta$, and

$$\mathbb{E}\left[\sup_x \sup_y T_{xy}\right] \leq \frac{1}{\theta} + o(1) \ .$$

The parameter θ can be tweaked to obtain an acceptable compromise between storage and routing time.

4 The Random ID Model

In this section, we assume that $X_1, \ldots, X_n$ are chosen uniformly at random from $\{0,1\}^d$ without replacement. Recall that $\widehat{X}_1$ denotes the ID that is farthest from X_1. We sketch the proof that the distribution of $\widehat{T} \overset{\text{def}}{=} T_{X_1 \widehat{X}_1}$ has concentrated mass.

Since rotating the hypercube does not change the distribution of the routing time, we can always assume that $X_1 = \bar{0}$ and $\widehat{X}_1 = \bar{1}$, where $\bar{b}$ denotes the all-b vector $(b, b, \ldots, b)$.

Write the routing path $\rho_{X_1 \widehat{X}_1} = (z_0, z_1, \ldots)$. Let a_t be the lowest common ancestor of z_t and y', i.e., the rightmost leaf in the trie and the destination of routing. Then the sequence $(a_0, a_1, \ldots)$ can be seen as travelling downwards along the path from the root to y', i.e., the rightmost branch of the trie, with the distance of each hop being random.

This sequence can be defined equivalently as follows. Let a_0 be the root of the ID trie. Let $z_0 = \bar{0}$. From the right subtree of node a_t, select up to k paths to the bottom uniformly at random without replacement. (This is equivalent to the choice of k nodes to fill one k-bucket of z_t, the t-th hop in the search.) If that right subtree is empty, then the search terminates at a_t. Let z_{t+1} be the leaf corresponding to the rightmost one of these selected paths. Let a_{t+1} be the lowest common ancestor of z_{t+1} and y'. Let L_{t+1} be the distance from a_0 to a_{t+1}. Let $R_{t+1} = L_{t+1} - L_t$, i.e., the distance of the $(t+1)$-th hop, which is also the distance between a_t and a_{t+1}. Note that $\widehat{T} = t$ if and only if $\sum_{i=1}^{t} R_i = d$. Therefore, we can bound $\widehat{T}$ by studying the properties of $(R_t)_{t \geq 1}$.

Now instead of the ID trie of depth d, consider a full binary trie with infinite depth. Definite $(a'_0, a'_1, \ldots)$, the counterpart of $(a_0, a_1, \ldots)$ in this infinite trie, as follows. Let a'_0 be the root of the infinite trie. From the right subtree of node a'_t, select exactly k infinite downwards paths uniformly at random. (Since the probability of selecting the same path more than once is zero, "without replacement" is not necessary anymore.) Let z'_{t+1} be the rightmost of these selected paths. Let a'_{t+1} be the lowest common node of z'_{t+1} and $\bar{1}$. Let L'_{t+1} be the distance from a'_0 to a'_{t+1}. Let $G_{t+1} = L'_{t+1} - L'_t$, i.e., the distance of $(t+1)$-th hop, which is also the distance between a'_t and a'_{t+1}.

When n is large (and thus d is large), the behavior of $(R_t)_{t \geq 1}$ and $(G_t)_{t \geq 1}$ are very similar. But it is easy to see that, since each subtree of the infinite trie has exactly the same structure, $(G_t)_{t \geq 1}$ is a sequence of i.i.d. random variables with distribution

$$\mathbb{P}\{G_1 \leq i\} = \left(1 - \frac{1}{2^i}\right)^k \qquad \text{for all } i \in \mathbb{N}. \tag{2}$$

In other words, $(G_t)_{t \geq 1}$ is much easier to analyze. (Note that when $k = 1$, G_1 is simply the geometric distribution.) And it is possible to couple the random variables G_t and R_t.

When the downwards travel reaches the depth of $\log_2 n$, the routing can not last much longer. In fact, in the ID trie, a subtree whose root has depth at least

$\log_2 n$ has only $o(\log n)$ leaves with high probability [17]. Therefore, we define

$$T_n = \min\left\{t : t \geq 1, \sum_{i=1}^{t} G_i \geq \log_2 n\right\}.$$

Lemma 4. *We have*

$$\mathbb{E}\left[\frac{T_n}{\log n}\right] \to \frac{1}{\log(2) \times \mathbb{E}\left[G_1\right]},$$

as $n \to \infty$, and also

$$\frac{T_n}{\log n} \to \frac{1}{\log(2) \times \mathbb{E}\left[G_1\right]} \qquad \text{in probability,}$$

as $n \to \infty$.

Then, by coupling, we can show Lemma 5:

Lemma 5. *We have*

$$\frac{\widehat{T} - T_n}{\log n} \to 0 \qquad \text{in probability,}$$

as $n \to \infty$.

We omit the proof of these two lemmas due to space limitations. For details, see [17]. For other proofs, see the appendix of this paper at `http://xingshicai.ca/kad.pdf` .

References

1. Davis, C., Fernandez, J., Neville, S., McHugh, J.: Sybil attacks as a mitigation strategy against the storm botnet. In: Proceedings of the 3rd International Conference on Malicious and Unwanted Software, Malware 2008, pp. 32–40 (2008)
2. Davis, C.R., Neville, S., Fernandez, J.M., Robert, J.-M., McHugh, J.: Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 461–480. Springer, Heidelberg (2008)
3. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
4. Schollmeier, R.: A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: Proceedings of 1st International Conference on Peer-to-Peer Computing, pp. 101–102 (2001)
5. Steinmetz, R., Wehrle, K. (eds.): Peer-to-Peer Systems and Applications. LNCS, vol. 3485. Springer, Heidelberg (2005)
6. Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Looking up data in P2P systems. Communications of the ACM 46(2), 43–48 (2003)

7. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
8. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. SIGCOMM Computer Communication Review 31(4), 161–172 (2001)
9. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Computer Communication Review 31(4), 149–160 (2001)
10. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22, 41–53 (2004)
11. Crosby, S.A., Wallach, D.S.: An analysis of BitTorrent's two Kademlia-based DHTs. Rice University, Houston, TX, USA (2007)
12. Steiner, M., En-Najjary, T., Biersack, E.W.: A global view of Kad. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC 2007, pp. 117–122. ACM, New York (2007)
13. Falkner, J., Piatek, M., John, J.P., Krishnamurthy, A., Anderson, T.: Profiling a million user DHT. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC 2007, pp. 129–134. ACM, New York (2007)
14. Steiner, M., En-Najjary, T., Biersack, E.W.: Exploiting Kad: possible uses and misuses. SIGCOMM Computer Communication Review 37(5), 65–70 (2007)
15. Fredkin, E.: Trie memory. Communications of the ACM 3(9), 490–499 (1960)
16. Szpankowski, W.: Average Case Analysis of Algorithms on Sequences. Wiley, Chichester (2011)
17. Cai, X.S.: A probabilistic analysis of kademlia networks. Master's thesis, McGill University (August 2012)
18. David, H., Nagaraja, H.: Order Statistics. Wiley, Hoboken (2003)
19. Johnson, N., Kotz, S., Balakrishnan, N.: Continuous Univariate Distributions, vol. 2. Wiley, Hoboken (1995)

Approximating the Generalized Minimum Manhattan Network Problem*

Aparna Das[1], Krzysztof Fleszar[2], Stephen Kobourov[1], Joachim Spoerhase[2], Sankar Veeramoni[1], and Alexander Wolff[2]

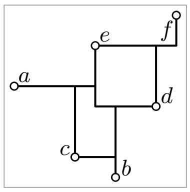
[1] Department of Computer Science, University of Arizona, Tucson, AZ, U.S.A.
[2] Lehrstuhl I, Institut für Informatik, Universität Würzburg, Germany

Abstract. We consider the *generalized minimum Manhattan network problem* (GMMN). The input to this problem is a set R of n pairs of terminals, which are points in $\mathbb{R}^2$. The goal is to find a minimum-length rectilinear network that connects every pair in R by a *Manhattan path*, that is, a path of axis-parallel line segments whose total length equals the pair's Manhattan distance. This problem is a natural generalization of the extensively studied *minimum Manhattan network problem* (MMN) in which R consists of all possible pairs of terminals. Another important special case is the well-known *rectilinear Steiner arborescence problem* (RSA). As a generalization of these problems, GMMN is NP-hard. No approximation algorithms are known for general GMMN.

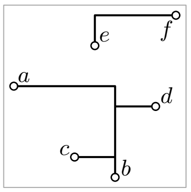
We obtain an $O(\log n)$-approximation algorithm for GMMN. Our solution is based on a stabbing technique, a novel way of attacking Manhattan network problems. Some parts of our algorithm generalize to higher dimensions, yielding a simple $O(\log^{d+1} n)$-approximation algorithm for the problem in arbitrary fixed dimension d. As a corollary, we obtain an exponential improvement upon the previously best $O(n^{\varepsilon})$-ratio for MMN in d dimensions [ESA'11]. En route, we show that an existing $O(\log n)$-approximation algorithm for 2D-RSA generalizes to higher dimensions.

1 Introduction

Given a set of terminals, which are points in $\mathbb{R}^2$, the *minimum Manhattan network problem* (MMN) asks for a minimum-length rectilinear network that connects every pair of terminals by a Manhattan path (*M-path*, for short), i.e., a path consisting of axis-parallel segments whose total length equals the pair's M-distance. Put differently, every pair is to be connected by a shortest path in the L_1-norm (M-path). See Fig. 1a for an example.



(a) an MMN for $\{a, b, c, d, e, f\}$



(b) a GMMN for $\{(a,b), (c,d), (e,f)\}$

Fig. 1: MMN versus GMMN

In the *generalized minimum Manhattan network problem* (GMMN), we are given a set R of n unordered terminal *pairs*, and

* This work was supported by the ESF EuroGIGA project GraDR (DFG grant Wo 758/5-1).

the goal is to find a minimum-length rectilinear network such that every pair in R is *M-connected*, that is, connected by an M-path. GMMN is a generalization of MMN since R may contain all possible pairs of terminals. Figure 1b depicts such a network.

We remark that, in this paper, we define n to be the number of terminal *pairs* of a GMMN instance, previous works on MMN defined n to be the number of *terminals*. Moreover, we identify each terminal pair with a rectangle, namely the bounding box of this pair. This is a natural convention as every M-path for this terminal pair lies within the bounding box.

MMN naturally arises in VLSI circuit layout [8], where a set of terminals (such as gates or transistors) needs to be interconnected by rectilinear paths (wires). Minimizing the cost of the network (which means minimizing the total wire length) is desirable in terms of energy consumption and signal interference. The additional requirement that the terminal pairs are connected by *shortest* rectilinear paths aims at decreasing the interconnection delay (see Cong et al. [4] for a discussion in the context of rectilinear Steiner arborescences, which have the same additional requirement; see definition below). Manhattan networks also arise in the area of geometric spanner networks. Specifically, a minimum Manhattan network can be thought of as the cheapest spanner under the L_1-norm for a given set of points (allowing Steiner points). Spanners, in turn, have numerous applications in network design, distributed algorithms, and approximation algorithms, see, e.g., the book [14] and the survey [9].

MMN requires a Manhattan path between every terminal pair. This assumption is, however, not always reasonable. For example, in VLSI design a wire connection is necessary only for an, often comparatively small, subset of terminal pairs, which may allow for substantially cheaper circuit layouts. In this scenario, GMMN appears to be a more realistic model than MMN.

Previous Work and Related Problems. MMN was introduced by Gudmundsson et al. [8] who gave 4- and 8-approximation algorithms for MMN running in $O(n^3)$ and $O(n \log n)$ time, respectively. The currently best known approximation algorithms for MMN have ratio 2; they were obtained independently by Chepoi et al. [2] using an LP-based method, by Nouioua [16] using a primal-dual scheme, and by Guo et al. [10] using a greedy approach. The complexity of MMN was settled only recently by Chin et al. [3]; they proved the problem NP-hard. It is not known whether MMN is APX-hard. Gudmundsson et al. [7] consider a variant of MMN where the goal is to minimize the number of (Steiner) nodes and edges. Using divide-and-conquer they show that there is always a Manhattan network with $O(n \log n)$ nodes and edges. Knauer and Spillner [11] show that MMN is fixed-parameter tractable. More specifically, they show that there is an exact algorithm for MMN taking $O^*(2^{14h})$ time, where h is the number of horizontal lines that contain all terminals and the O^*-notation neglects factors polynomial in n.

Recently, there has been an increased interest in MMN for higher dimensions. Muñoz et al. [13] proved that 3D-MMN is NP-hard to approximate within a factor of 1.00002. They also gave a constant-factor approximation algorithm for a (rather restricted) special case of 3D-MMN. Das et al. [6] described the first approximation algorithm for MMN in arbitrary, fixed dimension. Their algorithm recursively computes a grid and attaches the terminals within a grid cell to grid vertices using RSA as a subroutine. Its ratio is $O(n^\varepsilon)$ for any $\varepsilon > 0$.

GMMN was defined by Chepoi et al. [2] who posed the question whether it admits an $O(1)$-approximation. Suprisingly, only special cases of GMMN such as MMN have been considered so far—despite the fact that the problem is very natural and relevant for practical applications.

Another special case of GMMN that has received significant attention in the past is the *rectilinear Steiner arborescence problem* (RSA). Here, one is given a set of n terminals in the first quadrant, and the goal is to find a minimum-length rectilinear network that M-connects every terminal to the origin o. Hence, RSA is the special case of GMMN where o is considered a (new) terminal and the set of terminal pairs contains, for each terminal $t \neq o$, only the pair (o, t). Note that RSA is very different from MMN. Although every RSA solution is connected (via the origin), terminals are not necessarily *M-connected* to each other. RSA was introduced by Nastansky et al. [15]. RSA is NP-hard [18]. Rao et al. [17] gave a 2-approximation algorithm for RSA. They also provided a conceptually simpler $O(\log n)$-approximation algorithm based on rectilinear Steiner trees. In the full version of this paper [5], we generalize this algorithm to dimensions $d > 2$. Lu et al. [12] and, independently, Zachariasen [19] described polynomial-time approximation schemes (PTAS) for RSA, both based on Arora's technique [1]. Zachariasen pointed out that his PTAS can be generalized to the all-quadrant version of RSA but that it seems difficult to extend the approach to higher dimensions.

Our Contribution. Our main result is the first approximation algorithm for GMMN. Its ratio is $O(\log n)$ (see Section 3). Our algorithm is based on two ideas. First, we use a simple (yet powerful) divide-and-conquer scheme to reduce the problem to RSA. This yields a ratio of $O(\log^2 n)$. To bring down the ratio to $O(\log n)$ we develop a new *stabbing technique*, which is a novel way to approach Manhattan network problems and constitutes the main technical contribution of this paper.

We also consider higher dimensions. More specifically, we generalize an existing $O(\log n)$-approximation algorithm for RSA to arbitrary dimensions (see the full version [5]). Combining this with our divide-and-conquer scheme yields an $O(\log^{d+1} n)$-approximation algorithm for d-dimensional GMMN (see Section 4). For the special case of d-dimensional MMN, this constitutes an exponential improvement upon the $O(n^{\varepsilon})$-approximation algorithm of Das et al. [6]. Another advantage of our algorithm is that it is significantly simpler and easier to analyze than that algorithm.

Our result is a first step towards answering the open question of Chepoi et al. [2]. In the full version [5] we give indications that it may be difficult to obtain an $O(1)$-approximation algorithm since the problem can be viewed as a geometric rectangle covering problem. There we also argue why existing techniques for MMN seem to fail, which underlines the relevance of our techniques.

2 Divide-And-Conquer Scheme

As a warm-up, we start with a simple $O(\log^2 n)$-approximation algorithm illustrating our divide-and-conquer scheme. This is the basis for (a) an improved $O(\log n)$-approximation algorithm that uses our stabbing technique (see Section 3) and (b) a divide-and-conquer scheme for GMMN in arbitrary dimensions (Section 4). We prove the following.

Theorem 1. *GMMN admits an $O(\log^2 n)$-approximation algorithm running in $O(n \log^3 n)$ time.*

Our algorithm consists of a *main algorithm* that recursively subdivides the input instance into instances of so-called *x-separated* GMMN; see Section 2.1. We prove that the instances of x-separated GMMN can be solved independently by paying a factor of $O(\log n)$ in the overall approximation ratio. Then we solve each x-separated GMMN instance within factor $O(\log n)$; see Section 2.2. This yields an overall approximation ratio of $O(\log^2 n)$. Our analysis is tight; see the full version [5]. Our presentation follows this natural top-down approach; as a consequence, we will make some forward references to results that we prove later.

2.1 Main Algorithm

Our algorithm is based on divide and conquer. Let R be the set of terminal pairs that are to be M-connected. Recall that we identify each terminal pair with its bounding box. As a consequence of this, we consider R, a set of rectangles. Let m_x be the median in the multiset of the x-coordinates of terminals where a terminal occurs as often as the number of pairs it is involved in. We identify m_x with the vertical line at $x = m_x$.

Now we partition R into three subsets $R_{\mathrm{left}}, R_{\mathrm{mid}}$, and R_{right}. R_{left} consists of all rectangles that lie *completely* to the left of the vertical line m_x. Similarly, R_{right} consists of all rectangle that lie *completely* to the right of m_x. R_{mid} consists of all rectangles that intersect m_x.

We consider the sets R_{left}, R_{mid}, and R_{right} as separate instances of GMMN. We apply the main algorithm recursively to R_{left} to get a rectilinear network that M-connects terminal pairs in R_{left} and do the same for R_{right}.

It remains to M-connect the pairs in R_{mid}. We call a GMMN instance (such as R_{mid}) *x-separated* if there is a vertical line (in our case m_x) that intersects every rectangle. We exploit this property to design a simple $O(\log n)$-approximation algorithm for x-separated GMMN; see Section 2.2. In Section 3, we improve upon this and describe an $O(1)$-approximation algorithm for x-separated GMMN.

In the following lemma we analyze the performance of the main algorithm, in terms of $\rho_x(n)$, our approximation ratio for x-separated instances with n terminal pairs.

Lemma 1. *Let $\rho_x(n)$ be a non-decreasing function. Then, if x-separated GMMN admits a $\rho_x(n)$-approximation algorithm, GMMN admits a $(\rho_x(n) \cdot \log n)$-approximation algorithm.*

Proof. We determine an upper bound $\rho(n)$ on the main algorithm's approximation ratio for instances with n terminal pairs. Let N^{opt} be an optimum solution to an instance R of size n and let OPT be the cost of N^{opt}. Let $N^{\mathrm{opt}}_{\mathrm{left}}$ and $N^{\mathrm{opt}}_{\mathrm{right}}$ be the parts of N^{opt} to the left and to the right of m_x, respectively. (We split horizontal segments that cross m_x and ignore vertical segments on m_x.)

Due to the choice of m_x, at most n terminals lie to the left of m_x. Therefore, R_{left} contains at most $n/2$ terminal pairs. Since $N^{\mathrm{opt}}_{\mathrm{left}}$ is a feasible solution to R_{left}, we conclude (by induction) that the cost of the solution to R_{left} computed by our algorithm is bounded by $\rho(n/2) \cdot \|N^{\text{opt}}_{\text{left}}\|$, where $\| \cdot \|$ measures the length of a network. Analogously, the cost of the solution computed for R_{right} is bounded by $\rho(n/2) \cdot \|N^{\text{opt}}_{\text{right}}\|$. Since N^{opt} is also a feasible solution to the x-separated instance R_{mid}, we can compute a solution of cost $\rho_x(n) \cdot \text{OPT}$ for R_{mid}.

As the networks $N^{\text{opt}}_{\text{left}}$ and $N^{\text{opt}}_{\text{right}}$ are separated by line m_x, they are edge disjoint and hence $\|N^{\text{opt}}_{\text{left}}\| + \|N^{\text{opt}}_{\text{right}}\| \leq \text{OPT}$. Therefore, we can bound the total cost of our algorithm's solution N to R by

$$\rho(n/2) \cdot (\|N^{\text{opt}}_{\text{left}}\| + \|N^{\text{opt}}_{\text{right}}\|) + \rho_x(n) \cdot \text{OPT} \leq (\rho(n/2) + \rho_x(n)) \cdot \text{OPT} \ .$$

This yields the recurrence $\rho(n) = \rho(n/2) + \rho_x(n)$, which resolves to $\rho(n) \leq \log n \cdot \rho_x(n)$. □

Lemma 1 together with the results of Section 2.2 allow us to prove Theorem 1.

Proof (of Theorem 1). By Lemma 1, our main algorithm has performance $\rho_x(n) \cdot \log n$, where $\rho_x(n)$ denotes the ratio of an approximation algorithm for x-separated GMMN. In Lemma 2 (Section 2.2), we will show that there is an algorithm for x-separated GMMN with ratio $\rho_x(n) = O(\log n)$. Thus overall, the main algorithm yields an $O(\log^2 n)$-approximation for GMMN. See the full version [5] for the running time analysis. □

2.2 Approximating x-Separated and xy-Separated Instances

We describe a simple algorithm for approximating x-separated GMMN with a ratio of $O(\log n)$. Let R be an x-separated instance, that is, all rectangles in R intersect a common vertical line.

The algorithm works as follows. Analogously to the main algorithm we subdivide the x-separated input instance, but this time using the line $y = m_y$, where m_y is the median of the multiset of y-coordinates of terminals in R. This yields sets R_{top}, R'_{mid}, and R_{bottom}, defined analogously to the sets R_{left}, R_{mid}, and R_{right} of the main algorithm, using m_y instead of m_x. We apply our x-separated algorithm to R_{top} and then to R_{bottom} to solve them recursively. The instance R'_{mid} is a *y-separated* sub-instance with all its rectangles intersecting the line m_y. Moreover, R'_{mid} (as a subset of R) is already x-separated, thus we call R'_{mid} an *xy-separated* instance. Below, we describe a specialized algorithm to approximate xy-separated instances within a constant factor. Assuming this for now, we prove the following.

Lemma 2. *x-separated GMMN admits an $O(\log n)$-approximation algorithm.*

Proof. Let $\rho_x(n)$ be the ratio of our algorithm for approximating x-separated GMMN instances and let $\rho_{xy}(n)$ be the ratio of our algorithm for approximating xy-separated GMMN instances. In Lemma 3, we show that $\rho_{xy}(n) = O(1)$.

Following the proof of Lemma 1 (exchanging x- and y-coordinates and using R_{top}, R'_{mid}, R_{bottom} in place of $R_{\text{left}}, R_{\text{mid}}, R_{\text{right}}$), yields $\rho_x(n) = \log n \cdot \rho_{xy}(n) = O(\log n)$. □

It remains to show that xy-separated GMMN can be approximated within a constant ratio. Let R be an instance of xy-separated GMMN. We assume, w.l.o.g., that it is the x- and the y-axes that intersect all rectangles in R, that is, all rectangles contain the origin o. To solve R, we compute an RSA network that M-connects the set of terminals in R to o. Clearly, we obtain a feasible GMMN solution to R. In the full version [5] we prove that this is a constant-factor approximation algorithm.

Lemma 3. *xy-separated GMMN admits a constant-factor approximation algorithm.*

3 An $O(\log n)$-Approximation Algorithm via Stabbing

In this section, we present an $O(\log n)$-approximation algorithm for GMMN, which is the main result of our paper. Our algorithm relies on an $O(1)$-approximation algorithm for x-separated instances and is based on a novel stabbing technique that computes a cheap set of horizontal line segments that stabs all rectangles. Our algorithm connects these line segments with a suitable RSA solution to ensure feasibility and approximation ratio. We show the following (noting that our analysis is tight up to a constant factor; see the full version [5]).

Theorem 2. *For any $\varepsilon > 0$, GMMN admits a $((6+\varepsilon)\cdot\log n)$-approximation algorithm running in $O(n^{1/\varepsilon}\log^2 n)$ time.*

Proof. Using our new subroutine for the x-separated case given in Lemma 7 below, along with Lemma 1 yields the result. See the full version [5] for the run-time analysis. □

We begin with an overview of our improved algorithm for x-separated GMMN. Let R be the set of terminal pairs of an x-separated instance of GMMN. We assume, w.l.o.g., that each terminal pair $(t, t') \in R$ is separated by the y-axis, that is, $x(t) \leq 0 \leq x(t')$ or $x(t') \leq 0 \leq x(t)$. Let N^{opt} be an optimum solution to R. Let $\mathrm{OPT}_{\mathrm{ver}}$ and $\mathrm{OPT}_{\mathrm{hor}}$ be the total costs of the vertical and horizontal segments in N^{opt}, respectively. Hence, $\mathrm{OPT} = \mathrm{OPT}_{\mathrm{ver}} + \mathrm{OPT}_{\mathrm{hor}}$. We first compute a set S of horizontal line segments of total cost $O(\mathrm{OPT}_{\mathrm{hor}})$ such that each rectangle in R is *stabbed* by some line segment in S; see Sections 3.1 and 3.2. Then we M-connect the terminals to the y-axis so that the resulting network, along with S, forms a feasible solution to R of cost $O(\mathrm{OPT})$; see Section 3.3.

3.1 Stabbing the Right Part

We say that a horizontal line segment h *stabs* an axis-aligned rectangle r if the intersection of r and h equals the intersection of r and the supporting line of h. A set of horizontal line segments is a *stabbing* of a set of axis-aligned rectangles if each rectangle is stabbed by some line segment. For any geometric object, let its *right part* be its intersection with the closed half plane to the right of the y-axis. For a *set* of objects, let its right part be the set of the right parts of the objects. Let R^+ be the right part of R, let N^+ be the right part of N^{opt}, and let N^+_{hor} be the set of horizontal line segments in N^+. In this section, we show how to construct a stabbing of R^+ of cost at most $2 \cdot \|N^+_{\text{hor}}\|$.

For $x' \geq 0$, let $\ell_{x'}$ be the vertical line at $x = x'$. Our algorithm performs a left-to-right sweep starting with ℓ_0. For $x \geq 0$, let $\mathcal{I}_x = \{r \cap \ell_x \mid r \in R^+\}$ be the "traces" of the rectangles in R^+ on ℓ_x. The elements of $\mathcal{I}_x$ are vertical line segments; we refer to them as *intervals*. A set P_x of points on ℓ_x constitutes a *piercing* for $\mathcal{I}_x$, if every interval in $\mathcal{I}_x$ contains a point in P_x.

Our algorithm continuously moves the line ℓ_x from left to right starting with $x = 0$. In doing so, we maintain an inclusion-wise minimal piercing P_x of $\mathcal{I}_x$ in the following way: At $x = 0$, we start with an arbitrary minimal piercing P_0. (Note that we can even compute an optimum piercing.) We update P_x whenever $\mathcal{I}_x$ changes. Observe that with increasing x, the set $\mathcal{I}_x$ can only inclusion-wise decrease as all rectangles in R^+ touch the y-axis. Therefore, it suffices to update the piercing P_x only at *event points*; x is an event point if and only if x is the x-coordinate of a right edge of a rectangle in R^+. Let x' and x'' be consecutive event points. Let x be such that $x' < x \leq x''$. Note that $P_{x'}$ is a piercing for $\mathcal{I}_x$ since $\mathcal{I}_x \subset \mathcal{I}_{x'}$. The piercing $P_{x'}$ is, however, not necessarily *minimal* w.r.t. $\mathcal{I}_x$. When the sweep line passes x', we therefore have to drop some of the points in $P_{x'}$ in order to obtain a new minimal piercing. This can be done by iteratively removing points from $P_{x'}$ such that the resulting set still pierces $\mathcal{I}_x$. We stop at the last event point (afterwards, $\mathcal{I}_x = \emptyset$) and output the *traces* of the piercing points in P_x for $x \geq 0$ as our stabbing.

Note that with increasing x, our algorithm only *removes* points from P_x but never add points. Thus, the traces of P_x form horizontal line segments that touch the y-axis. These line segments form a stabbing of R^+; see the thick solid line segments in Fig. 2a. The following lemma is crucial to prove the overall cost of the stabbing.

Lemma 4. *For any $x \geq 0$, it holds that $|P_x| \leq 2 \cdot |\ell_x \cap N^+_{\text{hor}}|$.*

Proof. Since P_x is a minimal piercing, there exists, for every $p \in P_x$, a *witness* $I_p \in \mathcal{I}_x$ that is pierced by p but not by $P_x \setminus \{p\}$. Otherwise we could remove p from P_x, contradicting the minimality of P_x.

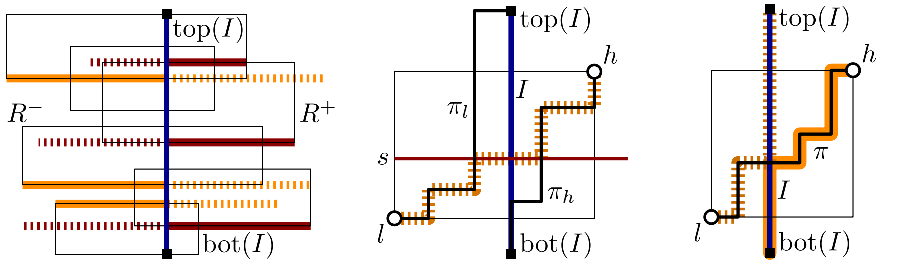
Now we show that an arbitrary point q on ℓ_x is contained in the witnesses of at most two points in P_x. Assume, for the sake of contradiction, that q is contained in the witnesses of points $p, p', p'' \in P_x$ with strictly increasing y-coordinates. Suppose that q lies above p'. Then the witness I_p of p, which contains p and q, must also contain p', contradicting the definition of I_p. The case q below p' is symmetric.

Observe that $\ell_x \cap N^+_{\text{hor}}$ is a piercing of $\mathcal{I}_x$ and, hence, of the $|P_x|$ many witnesses. Since every point in $\ell_x \cap N^+_{\text{hor}}$ pierces at most two witnesses, the lemma follows. □

Next, we analyze the overall cost of the stabbing.

Lemma 5. *Given a set R of rectangles intersecting the y-axis, we can compute a set of horizontal line segments of cost at most $2 \cdot \text{OPT}_{\text{hor}}$ that stabs R^+.*

Proof. Observe that $\|N^+_{\text{hor}}\| = \int |\ell_x \cap N^+_{\text{hor}}|\, dx$. The cost of our stabbing is $\int |P_x|\, dx$. By Lemma 4, this can be bounded by $\int |P_x|\, dx \leq \int 2 \cdot |\ell_x \cap N^+_{\text{hor}}|\, dx = 2 \cdot \|N^+_{\text{hor}}\|$. □



(a) The dark (light) segments S^+ (S^-) stab R^+ (R^-). The dotted segments are mirror images of $S^+ \cup S^-$.

(b) $N = A_{\text{up}} \cup A_{\text{down}} \cup S$ is feasible for R.

(c) $N^{\text{opt}} \cup \{I\}$ is feasible for RSA instances $(L, \text{top}(I))$, $(H, \text{bot}(I))$.

Fig. 2: The improved algorithm for x-separated GMMN

3.2 Stabbing the Right and Left Parts

We now detail how we construct a stabbing of R. To this end we apply Lemma 5 to compute a stabbing S^- of cost at most $2 \cdot \|N^-_{\text{hor}}\|$ for the left part R^- of R and a stabbing S^+ of cost at most $2 \cdot \|N^+_{\text{hor}}\|$ for the right part R^+. Note that $S^- \cup S^+$ is not necessarily a stabbing of R since there can be rectangles that are not *completely* stabbed by one segment (even if we start with the same piercing on the y-axis in the sweeps to the left and to the right). To overcome this difficulty, we mirror S^- and S^+ to the respective other side of the y-axis; see Fig. 2a. Let S denote the union of $S^- \cup S^+$ and the mirror image of $S^- \cup S^+$.

Lemma 6. *Given a set R of rectangles intersecting the y-axis, we can compute a set of horizontal line segments of cost at most $4 \cdot \text{OPT}_{\text{hor}}$ that stabs R.*

Proof. Let S be the set of horizontal line segments described above. The total cost of S is at most $4(\|N^-_{\text{hor}}\| + \|N^+_{\text{hor}}\|) = 4 \cdot \text{OPT}_{\text{hor}}$. The set S stabs R since, for every rectangle $r \in R$, the larger among its two (left and right) parts is stabbed by some segment s and the smaller part is stabbed by the mirror image s' of s. Hence, r is stabbed by the line segment $s \cup s'$. □

3.3 Connecting Terminals and Stabbing

We assume that the union of the rectangles in R is connected. Otherwise we apply our algorithm separately to each subset of R that induces a connected component of $\bigcup R$. Let I be the line segment that is the intersection of the y-axis with $\bigcup R$. Let $\text{top}(I)$ and $\text{bot}(I)$ be the top and bottom endpoints of I, respectively. Let $L \subseteq T$ be the set containing every terminal t with $(t, t') \in R$ and $y(t) \leq y(t')$ for some $t' \in T$. Symmetrically, let $H \subseteq T$ be the set containing every terminal t with $(t, t') \in R$ and $y(t) \geq y(t')$ for some $t' \in T$. Note that, in general, L and H are not disjoint.

Using a PTAS for RSA [12,19], we compute a near-optimal RSA network A_{up} connecting the terminals in L to $\text{top}(I)$ and a near-optimal RSA network A_{down} connecting the terminals in H to $\mathrm{bot}(I)$. Then we return the network $N = A_{\mathrm{up}} \cup A_{\mathrm{down}} \cup S$, where S is the stabbing computed by the algorithm in Section 3.2.

We prove in the following lemma that the resulting network is a feasible solution to R, with cost at most constant times OPT.

Lemma 7. *x-separated GMMN admits, for any $\varepsilon > 0$, a $(6 + \varepsilon)$-approximation algorithm.*

Proof. First we argue that the solution is feasible. Let $(l, h) \in R$. W.l.o.g., $y(l) \le y(h)$ and thus $l \in L$ and $h \in H$. Hence, A_{up} contains a path π_l from l to $\mathrm{top}(I)$, see Fig. 2b. This path starts inside the rectangle (l, h). Before leaving (l, h), the path intersects a line segment s in S that stabs (l, h). The segment s is also intersected by the path π_h in A_{down} that connects h to $\mathrm{bot}(I)$. Hence, walking along π_l, s, and π_h brings us in a monotone fashion from l to h.

Now, let us analyze the cost of N. Clearly, the projection of (the vertical line segments of) N^{opt} onto the y-axis yields the line segment I. Hence, $\|I\| \le \mathrm{OPT}_{\mathrm{ver}}$. Observe that $N^{\mathrm{opt}} \cup \{I\}$ constitutes a solution to the RSA instance $(L, \mathrm{top}(I))$ connecting all terminals in L to $\mathrm{top}(I)$ and to the RSA instance $(H, \mathrm{bot}(I))$ connecting all terminals in H to $\mathrm{bot}(I)$. This holds since, for each terminal pair, its M-path π in N^{opt} crosses the y-axis in I; see Fig. 2c. Since A_{up} and A_{down} are near-optimal solutions to these RSA instances, we obtain, for any $\delta > 0$, that $\|A_{\mathrm{up}}\| \le (1 + \delta) \cdot \|N^{\mathrm{opt}} \cup I\| \le (1+\delta) \cdot (\mathrm{OPT} + \mathrm{OPT}_{\mathrm{ver}})$ and, analogously, that $\|A_{\mathrm{down}}\| \le (1+\delta) \cdot (\mathrm{OPT} + \mathrm{OPT}_{\mathrm{ver}})$.

By Lemma 6, we have $\|S\| \le 4 \cdot \mathrm{OPT}_{\mathrm{hor}}$. Assuming $\delta \le 1$, this yields

$$\begin{aligned} \|N\| &= \|A_{\mathrm{up}}\| + \|A_{\mathrm{down}}\| + \|S\| \le (2 + 2\delta) \cdot (\mathrm{OPT} + \mathrm{OPT}_{\mathrm{ver}}) + 4 \cdot \mathrm{OPT}_{\mathrm{hor}} \\ &\le (2 + 2\delta) \cdot \mathrm{OPT} + 4 \cdot (\mathrm{OPT}_{\mathrm{ver}} + \mathrm{OPT}_{\mathrm{hor}}) = (6 + 2\delta) \cdot \mathrm{OPT} \ . \end{aligned}$$

Setting $\delta = \varepsilon/2$ yields the desired approximation factor. □

4 Generalization to Higher Dimensions

In this section, we describe an $O(\log^{d+1} n)$-approximation algorithm for GMMN in d dimensions and prove the following result (see below for the proof). In the full version [5] we show that the analysis of the algorithm is essentially tight (up to one log-factor).

Theorem 3. *In any fixed dimension d, GMMN admits an $O(\log^{d+1} n)$-approximation algorithm running in $O(n^2 \log^{d+1} n)$ time.*

In Section 2 we reduced GMMN to x-separated GMMN and then x-separated GMMN to xy-separated GMMN. Each of the two reductions increased the approximation ratio by a factor of $O(\log n)$. The special case of xy-separated GMMN was approximated within a constant factor by solving a related RSA problem. This gave an overall $O(\log^2 n)$-approximation algorithm for GMMN. We generalize this approach to higher dimensions.

An instance R of d-dimensional GMMN is called *j-separated* for some $j \le d$ if there exist values $s_1, \ldots, s_j$ such that, for each terminal pair $(t, t') \in R$ and for each dimension $i \leq j$, we have that s_i *separates* the i-th coordinates $x_i(t)$ of t and $x_i(t')$ of t' (meaning that either $x_i(t) \leq s_i \leq x_i(t')$ or $x_i(t') \leq s_i \leq x_i(t)$). Under this terminology, an arbitrary instance of d-dimensional GMMN is always *0-separated*.

The following lemma reduces j-separated GMMN to $(j-1)$-separated GMMN at the expense of a $(\log n)$-factor in the approximation ratio. The proof is similar to the 2D case; see the full version [5].

Lemma 8. *Let $1 \leq j \leq d$. If j-separated GMMN admits a $\rho_j(n)$-approximation algorithm, then $(j-1)$-separated GMMN admits a $(\rho_j(n) \cdot \log n)$-approximation algorithm.*

Analogously to dimension two we can approximate instances of d-separated GMMN by reducing the problem to RSA. Rao et al. [17] presented an $O(\log |T|)$-approximation algorithm for 2D-RSA, which generalizes to d-dimensional RSA as we show in the full version [5]. Using this, we derive there the following result.

Lemma 9. *d-separated GMMN admits an $O(\log n)$-approximation algorithm for any fixed dimension d.*

We are now ready to give the proof of Theorem 3.

Proof (Proof of Theorem 3). Combining Lemmata 8 and 9 and applying them inductively to arbitrary (that is, 0-separated) GMMN instances yields the claim. See the full version [5] for the run-time analysis. □

As a byproduct of Theorem 3, we obtain an $O(\log^{d+1} n)$-approximation algorithm for *MMN* where n denotes the number of *terminals*. This holds since any MMN instance with n terminals can be considered an instance of GMMN with $O(n^2)$ terminal pairs.

Corollary 1. *In any fixed dimension d, MMN admits an $O(\log^{d+1} n)$-approximation algorithm running in $O(n^4 \log^{d+1} n)$ time, where n denotes the number of terminals.*

5 Conclusions

In 2D, there is quite a large gap between the currently best approximation ratios for MMN and GMMN. Whereas we have presented an $O(\log n)$-approximation algorithm for GMMN, MMN admits 2-approximation algorithms [2,10,16]. In the full version [5], we give indications that this gap might not only be a shortcoming of our algorithm. It would be interesting to derive *some* non-approximability result for GMMN. So far, the only such result is the APX-hardness of 3D-MMN [13].

Concerning the positive side, for $d \geq 3$, a constant-factor approximation algorithm for d-dimensional RSA would shave off a factor of $O(\log n)$ from the current ratio for d-dimensional GMMN. This may be in reach since 2D-RSA admits even a PTAS [12,19]. Alternatively, a constant-factor approximation algorithm for $(d-k)$-separated GMMN for some $k \leq d$ would shave off a factor of $O(\log^k n)$ from the current ratio for d-dimensional GMMN.

Acknowledgments. We thank Michael Kaufmann for his hospitality and his enthusiasm during our respective stays in Tübingen. We thank Esther Arkin, Alon Efrat, Joe Mitchell, and Andreas Spillner for discussions.

References

1. Arora, S.: Approximation schemes for NP-hard geometric optimization problems: A survey. Math. Program. 97(1-2), 43–69 (2003)
2. Chepoi, V., Nouioua, K., Vaxès, Y.: A rounding algorithm for approximating minimum Manhattan networks. Theor. Comput. Sci. 390(1), 56–69 (2008)
3. Chin, F., Guo, Z., Sun, H.: Minimum Manhattan network is NP-complete. Discrete Comput. Geom. 45, 701–722 (2011)
4. Cong, J., Leung, K.S., Zhou, D.: Performance-driven interconnect design based on distributed RC delay model. In: 30th IEEE Conf. Design Automation (DAC 1993), pp. 606–611. IEEE Press, New York (1993)
5. Das, A., Fleszar, K., Kobourov, S.G., Spoerhase, J., Veeramoni, S., Wolff, A.: Approximating the generalized minimum Manhattan network problem. Arxiv report (2012), `http://arxiv.org/abs/1203.6481`
6. Das, A., Gansner, E.R., Kaufmann, M., Kobourov, S., Spoerhase, J., Wolff, A.: Approximating minimum Manhattan networks in higher dimensions. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 49–60. Springer, Heidelberg (2011), to appear in Algorithmica, `http://dx.doi.org/10.1007/s00453-013-9778-z`
7. Gudmundsson, J., Klein, O., Knauer, C., Smid, M.: Small Manhattan networks and algorithmic applications for the Earth Mover's Distance. In: 23rd Europ. Workshop Comput. Geom (EuroCG 2007), Graz, Austria, pp. 174–177 (2007)
8. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Approximating a minimum Manhattan network. Nordic J. Comput. 8, 219–232 (2001)
9. Gudmundsson, J., Narasimhan, G., Smid, M.: Applications of geometric spanner networks. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, pp. 1–99. Springer (2008)
10. Guo, Z., Sun, H., Zhu, H.: Greedy construction of 2-approximate minimum Manhattan networks. Int. J. Comput. Geom. Appl. 21(3), 331–350 (2011)
11. Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum Manhattan network problem. J. Comput. Geom. 2(1), 189–204 (2011)
12. Lu, B., Ruan, L.: Polynomial time approximation scheme for the rectilinear Steiner arborescence problem. J. Comb. Optim. 4(3), 357–363 (2000)
13. Muñoz, X., Seibert, S., Unger, W.: The minimal Manhattan network problem in three dimensions. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 369–380. Springer, Heidelberg (2009)
14. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)
15. Nastansky, L., Selkow, S.M., Stewart, N.F.: Cost-minimal trees in directed acyclic graphs. Zeitschrift Oper. Res. 18(1), 59–67 (1974)
16. Nouioua, K.: Enveloppes de Pareto et Réseaux de Manhattan: Caractérisations et Algorithmes. PhD thesis, Université de la Méditerranée (2005), `http://www.lif-sud.univ-mrs.fr~karim/download/THESE_NOUIOUA.pdf`
17. Rao, S., Sadayappan, P., Hwang, F., Shor, P.: The rectilinear Steiner arborescence problem. Algorithmica 7, 277–288 (1992)
18. Shi, W., Su, C.: The rectilinear Steiner arborescence problem is NP-complete. SIAM J. Comput. 35(3), 729–740 (2005)
19. Zachariasen, M.: On the approximation of the rectilinear Steiner arborescence problem in the plane (2000) (Manuscript), `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.4529`

Minmax Regret 1-Facility Location on Uncertain Path Networks

Haitao Wang

Department of Computer Science
Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract. Let P be an undirected path graph of n vertices. Each edge of P has a positive length and a constant capacity. Every vertex has a nonnegative supply, which is an unknown value but is known to be in a given interval. The goal is to find a point on P to build a facility and move all vertex supplies to the facility such that the maximum regret is minimized. The previous best algorithm solves the problem in $O(n \log^2 n)$ time and $O(n \log n)$ space. In this paper, we present an $O(n \log n)$ time and $O(n)$ space algorithm, and our approach is based on new observations and algorithmic techniques.

1 Introduction

Facility location problems on networks have received considerable attention over a few decades. The problems are normally concerned with networks where the information (e.g., the vertex and the edge weights) are known precisely. However, data in practice often involve uncertainty and may change with the time. Recently facility locations problems in uncertain environments have been studied, e.g., [1–11, 13, 16, 17]. One approach that is often used to model the uncertainty is the *worst-case analysis* in which one is looking for a solution that performs reasonably well for all possible *scenarios* (where a scenario is a specific realization of all uncertain parameters of the problem). There are many optimization criteria in the worst-case analysis. In particular, the *minmax regret optimization* aims at obtaining a solution that minimizes the maximum deviation, over all possible scenarios, between the value of the solution and the optimal value of the corresponding scenario, e.g., [2, 4, 6–8, 10, 13, 17]. In other words, the minmax regret optimization seeks to minimize the worst-case loss in the objective function value that may occur because the solution is chosen without knowing which scenario will take place.

In this paper, we consider the *minmax regret* 1*-facility location problem on uncertain path networks* where the vertex weights are uncertain. The problem was proposed recently by Cheng *et al.* [10] and an $O(n \log^2 n)$ time and $O(n \log n)$ space algorithm was given in [10]. By discovering more observations, we present an $O(n \log n)$ time and $O(n)$ space algorithm. As discussed in [10], the problem is motivated by an earthquake evacuation problem due to the Tohoku-Pacific

Ocean Earthquake that happened in Japan on March 11st, 2011. For example, suppose we have a highway that connects many cities and we want to find a location on the highway to build an evacuation facility such that when earthquake happens we can evacuate people in all these cities to the facility as soon as possible. The number of people in each city is uncertain due to different time (e.g., weekdays, weekends, days, nights, holidays). We formally introduce the problem below, and some notations are borrowed from [10].

1.1 Problem Definitions

Let $P = (V, E)$ be a path graph, with the vertex set $V = \{v_1, \ldots, v_n\}$ and the edge set $E = \{e_1, \ldots, e_{n-1}\}$, such that e_i connects v_i and v_{i+1} for each $1 \le i \le n-1$. Each edge $e \in E$ has a positive weight $l(e)$. Each vertex $v_i \in V$ has a weight w_i (e.g., the number of evacuees), which is unknown but is known in a given interval $[w_i^-, w_i^+]$ with $0 \le w_i^- \le w_i^+$. Let c be a constant representing the *capacity* of each edge, which is the maximum number of evacuees passing any point in any unit time. Let τ be a positive constant representing the time required for traversing a unit distance of every evacuee. Let Σ be the Cartesian product of all intervals $[w_i^-, w_i^+]$ for $1 \le i \le n$. Every element $s \in \Sigma$ is called a *scenario* that is a feasible assignment of weights to the vertices of P. For any scenario $s \in \Sigma$, for each $1 \le i \le n$, we denote by $w_i(s)$ the weight of the vertex v_i in the scenario s, and $w_i^- \le w_i(s) \le w_i^+$.

As in [10], we embed the path P on a real line L (e.g., the x-axis) such that each vertex $v_i \in V$ is associated with the coordinate $x_i = x_1 + \sum_{j=1}^{i-1} l(e_j)$ for each $2 \le i \le n$. For each point $x \in L$, with a little abuse of notation, we use x to denote the coordinate of the point. We also use P to denote the set of points x on L with $x_1 \le x \le x_n$. For any point $x \in P$, let $P_L(x) = \{t \in P \mid t < x\}$ and $P_R(x) = \{t \in P \mid t > x\}$. Suppose we build a facility at a location $x \in P$. Consider any scenario $s \in \Sigma$. We use $T_L(x, s)$ to denote the minimum time for the evacuees on $P_L(x)$ to move to x; similarly, let $T_R(x, s)$ denote the minimum time for the evacuees on $P_R(x)$ to move to x. Note that if x is at a vertex $v_i \in V$, then we assume the evacuees at v_i can complete evacuation in no time. As discussed in [10], by [12], $T_L(x, s)$ and $T_R(x, s)$ can be expressed as: $T_L(x, s) = \max_{v_i \in P_L(x)}\{(x - x_i) \cdot \tau + \lceil \frac{1}{c} \cdot \sum_{j=1}^{i} w_j(s) \rceil - 1\}$, and $T_R(x, s) = \max_{v_i \in P_R(x)}\{(x_i - x) \cdot \tau + \lceil \frac{1}{c} \cdot \sum_{j=i}^{n} w_j(s) \rceil - 1\}$. Note that if $P_L(x) = \emptyset$, $T_L(x, s) = 0$, and if $P_R(x) = \emptyset$, $T_R(x, s) = 0$.

As in [10], we only need to consider the special case $c = 1$ since the general case can be treated in the similar way, and also, we can omit -1 from the formulas above when designing the algorithm. Hence, as in [10], we can simply use the following definitions: $T_L(x, s) = \max_{v_i \in P_L(x)}\{(x - x_i) \cdot \tau + \sum_{j=1}^{i} w_j(s)\}$, and $T_R(x, s) = \max_{v_i \in P_R(x)}\{(x_i - x) \cdot \tau + \sum_{j=i}^{n} w_j(s)\}$.

As in [10], for convenience of discussion, for each $1 \le i \le n$, we define a function $f_L^i(x, s)$ on $x > x_i$ and a function $f_R^i(s, x)$ on $x < x_i$ as follows: $f_L^i(x, s) = (x - x_i) \cdot \tau + \sum_{j=1}^{i} w_j(s)$ and $f_R^i(x, s) = (x_i - x) \cdot \tau + \sum_{j=i}^{n} w_j(s)$. Hence, we have $T_L(x, s) = \max_{v_i \in P_L(x)} f_L^i(x, s)$ and $T_R(x, s) = \max_{v_i \in P_R(x)} f_R^i(x, s)$.

Let $T(x,s)$ to denote the minimum time for all evacuees on P to move to x. Thus, $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$. Denote by $x_{opt}(s)$ a point on P such that $T(x,s)$ is minimized when $x = x_{opt}(s)$, and one may consider $x_{opt}(s)$ as an *optimal location* for the scenario s. For any point x on L, let $R(x,s) = T(x,s) - T(x_{opt}(s),s)$, and we call $R(x,s)$ the *regret* of x in the scenario s. Intuitively, $R(x,s)$ is the regret (i.e., the opportunity loss) caused by choosing the location x instead of the optimal location $x_{opt}(s)$. Finally, the *maximum regret* of x is defined as $R_{max}(x) = \max_{s\in\Sigma} R(x,s)$. In other words, $R_{max}(x)$ is the worst-case opportunity loss for choosing the location x.

Our *minmax regret* problem is to choose a location x on L such that the maximum regret $R_{max}(x)$ is minimized, and the minimized $R_{max}(x)$ is called the *minmax regret.*

1.2 Our Results

We present an algorithm of $O(n \log n)$ time and $O(n)$ space for the problem, which improves the $O(n \log^2 n)$ time and $O(n \log n)$ space algorithm [10].

Our algorithm makes use of the critical observation given in [10] that there are a set S of $2n$ scenarios such that for any point x on L, the "worst-case" scenario for $R_{max}(x)$ must be in S. This implies that instead of considering the infinitely many scenarios of Σ for computing $R_{max}(x)$, we only need to consider the scenarios in S. The algorithm has two main steps. The first step is to compute the optimal positions for all scenarios in S. The second step is to compute the minmax regret. Algorithms of $O(n \log^2 n)$ time are given in [10] for each step. By discovering more observations, we solve each step in $O(n \log n)$ time (and $O(n)$ space) by even simpler algorithms. In particular, the high level scheme of our approach for the second step is binary search, whose efficiency hinges on solving the following sub-problem in linear time: Given any point x on L, compute the values $T_L(x,s)$ and $T_R(x,s)$ for all scenarios $s \in S$. A straightforward method can compute $T_L(x,s)$ and $T_R(x,s)$ in $O(n)$ time for each scenario s, and thus solves the sub-problem in $O(n^2)$ time. We present an $O(n)$ time algorithm for the sub-problem, and it should be noted that our algorithm itself is very simple but it is more challenging to observe the crucial properties behind the scene.

We discuss basic observations in Section 2. In Section 3, we compute the optimal locations for all scenarios in S. Section 4 computes the minmax regret.

2 Preliminaries

We discuss some basic observations. Most of them have been discovered in [10] and we sketch them in this section for completeness of this paper.

Our goal is to find a location x to minimize the maximum regret $R_{max}(x) = \max_{s\in\Sigma} R(x,s)$. Consider any point x on P and any scenario $s \in \Sigma$. To compute $R(x,s)$, we need to known $x_{opt}(s)$ first. Recall that $x_{opt}(s)$ is the value such that $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$ is minimized when $x = x_{opt}(s)$. To determine $x_{opt}(s)$, we discuss some properties of $T_L(x,s)$ and $T_R(x,s)$.

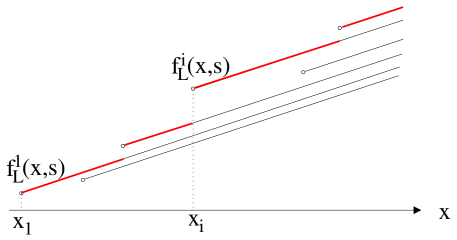


Fig. 1. Illustrating the functions $f_L^i(x,s)$. $T_L(x,s)$ is the upper envelope of them, shown with red color.

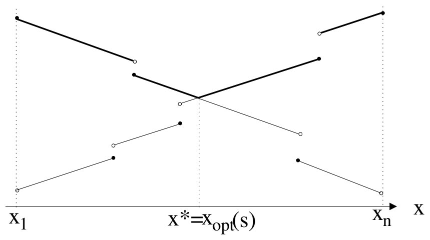


Fig. 2. Illustrating the function $T(x,s)$ shown with thick segments and the optimal location $x_{opt}(s)$

Recall that $T_L(x,s) = \max_{v_i \in P_L(x)} f_L^i(x,s)$. For each $1 \le i \le n$, the function $f_L^i(x,s)$ defines in the plane an open half-line of slope τ with (but excluding) the (left) endpoint $(x_i, \sum_{j=1}^{i} w_j(s))$ (e.g., see Fig. 1). $T_L(x,s)$ is the upper envelope of the n half-lines defined by the functions $f_L^i(x,s)$ for $i = 1,\ldots,n$. Since $\tau > 0$, $T_L(x,s)$ is a strictly increasing function of x (e.g., see Fig. 1). Similarly, each $f_R^i(x,s)$ defines an open half-line of slope $-\tau$ with (but excluding) the (right) endpoint $(x_i, \sum_{j=i}^{n} w_j(s))$, and $T_R(x,s)$ is the corresponding upper envelope, which is strictly decreasing. Since $T(x,s) = \max\{T_L(x,s), T_R(x,s)\}$, $T(x,s)$ is convex and there exists a value x^* such that $T(x,s)$ is strictly decreasing on $(-\infty, x^*]$ and increasing on $[x^*, +\infty)$ (e.g., see Fig. 2). Note that the above x^* is $x_{opt}(s)$. These properties are already given in [10]. The above discussion also shows that for any scenario $s \in \Sigma$, $T(x,s)$ is the upper envelope of the functions $f_L^i(x,s)$ and $f_R^i(x,s)$ for $i = 1,\ldots,n$.

For any point x, to compute the maximum regret $R_{max}(x)$, a straightforward approach is the enumerate all scenarios in Σ to compute $R(x,s)$ for every scenario $s \in \Sigma$. However, since there are infinitely many scenarios in Σ, the approach does not work. Below, we use a difference approach.

A scenario s is the *worst-case scenario* for the location x if $R_{max}(x) = R(x,s)$, and we denote it by $s(x)$. Clearly, if we know $s(x)$, then we can compute $R_{max}(x) = R(x, s(x))$. Cheng *et al.* [10] provided a way to determine a set S of at most $2n$ scenarios such that $s(x)$ must be in S for any x, as follows.

For each $1 \le i \le n$, let s_L^i be the scenario where the weight $w_j(s_L^i)$ of the vertex v_j is w_j^+ for each j with $1 \le j \le i$, and $w_j(s_L^i) = w_j^-$ for each j with $i+1 \le j \le n$ if $i < n$. Symmetrically, for each $1 \le i \le n$, let s_R^i be the scenario where $w_j(s_R^i) = w_j^-$ for each j with $1 \le j \le i$, and $w_j(s_R^i) = w_j^+$ for each j with $i+1 \le j \le n$ if $i < n$. Let $S_L = \{s_L^i \mid 1 \le i \le n\}$ and $S_R = \{s_R^i \mid 1 \le i \le n\}$. Let $S = S_L \cup S_R$. The following lemma has been proved in [10].

Lemma 1. [10] *For any point x on L, there exists a worst-case scenario for x in S.*

In light of Lemma 1, we have $R_{max}(x) = \max_{s \in S} R(x,s)$. Hence, to compute $R_{max}(x)$, instead of considering all scenarios of Σ, we only need to consider

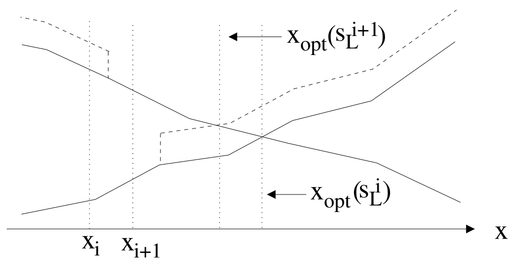


Fig. 3. Illustrating $T_L(s_L^i)$, $T_R(s_L^i)$, $T_L(s_L^{i+1})$, and $T_R(s_L^{i+1})$. $T_L(s_L^{i+1})$ (resp., $T_R(s_L^{i+1})$) can be obtained by shifting a portion of $T_L(s_L^i)$ (resp., $T_R(s_L^i)$) on the right (resp., left) of x_{i+1} upwards for $w_{i+1}^+ - w_{i+1}^-$

the $2n$ scenarios in S. For each $s \in S$, to compute $R(x, s)$, we need to know the optimal location $x_{opt}(s)$. Cheng *et al.* [10] presented an $O(n \log^2 n)$ time algorithm for computing $x_{opt}(s)$ for all scenarios $s \in S$, and in Section 3 we describe an $O(n \log n)$ time algorithm.

3 The Optimal Solutions for the Scenarios of *S*

In this section, we present an $O(n \log n)$ time and $O(n)$ space algorithm for computing $x_{opt}(s)$ for all scenarios $s \in S$, which improves the $O(n \log^2 n)$ time algorithm in [10]. Our improvement is due in a large part to certain monotone properties of the values $x_{opt}(s)$ given in the following lemma.

Lemma 2. *For any two scenarios s_L^i and s_L^{i+1} of S_L with $1 \leq i \leq n-1$, if $x_{i+1} \leq x_{opt}(s_L^i)$, then $x_{i+1} \leq x_{opt}(s_L^{i+1}) \leq x_{opt}(s_L^i)$; otherwise, $x_{opt}(s_L^i) \leq x_{opt}(s_L^{i+1}) \leq x_{i+1}$.*

Proof. We only prove the case where $x_{i+1} \leq x_{opt}(s_L^i)$ since the proof for the other case where $x_{i+1} > x_{opt}(s_L^i)$ is very similar.

According to the definitions of the two scenarios s_L^i and s_L^{i+1}, for each vertex v_j, if $j \neq i+1$, the weights of v_j in the two scenarios are the same, but for the vertex v_{i+1}, $w_{i+1}(s_L^i) = w_{i+1}^-$ and $w_{i+1}(s_L^{i+1}) = w_{i+1}^+$. By Corollary 1 in [10], $x_{i+1} \leq x_{opt}(s_L^{i+1})$ holds. Below, we prove $x_{opt}(s_L^{i+1}) \leq x_{opt}(s_L^i)$. To this end, it is sufficient to show that $T_L(x, s_L^{i+1}) > T_R(x, s_L^{i+1})$ for any $x > x_{opt}(s_L^i)$. The details are given below.

Consider any value $x > x_{opt}(s_L^i)$. Since $T_L(x, s)$ is strictly monotone increasing and $T_R(x, s)$ is strictly monotone decreasing for any scenario s, according to the definition of $x_{opt}(s_L^i)$, we have $T_L(x, s_L^i) > T_R(x, s_L^i)$.

According to the definitions of s_L^i and s_L^{i+1}, $f_L^j(t, s_L^{i+1}) \geq f_L^j(t, s_L^i)$ for any $j \geq i+1$ and any $t > x_j$ (more precisely, $f_L^j(t, s_L^{i+1}) = f_L^j(t, s_L^i) + w_{i+1}^+ - w_{i+1}^-$), and $f_L^j(t, s_L^{i+1}) = f_L^j(t, s_L^i)$ for any $j \leq i$ and any $t > x_j$ (e.g., see Fig. 3). Due to $x > x_{opt}(s_L^i) \geq x_{i+1}$, we obtain $T_L(x, s_L^{i+1}) \geq T_L(x, s_L^i)$.

Similarly, $f_R^j(t, s_L^{i+1}) \geq f_R^j(t, s_L^i)$ for any $j \leq i+1$ and any $t < x_j$, and $f_R^j(t, s_L^{i+1}) = f_R^j(t, s_L^i)$ for any $j \geq i+2$ and any $t < x_j$ (e.g., see Fig. 3). Since $x > x_{opt}(s_L^i) \geq x_{i+1}$, none of the functions $f_R^j(t, s_L^{i+1})$ for $j \leq i+1$ is defined on $t = x$. Therefore, we obtain $T_R(x, s_L^{i+1}) = T_R(x, s_L^i)$.

The above shows that $T_L(x, s_L^i) > T_R(x, s_L^i)$, $T_L(x, s_L^{i+1}) \geq T_L(x, s_L^i)$, and $T_R(x, s_L^{i+1}) = T_R(x, s_L^i)$. Hence, we conclude that $T_L(x, s_L^{i+1}) > T_R(x, s_L^{i+1})$.

Lemma 2 implies the following monotone property of $x_{opt}(s_L^i)$. Suppose initially $x_2 \leq x_{opt}(s_L^1)$; as the index i increases, $x_{opt}(x_L^i)$ moves monotonically "backward" to the left until at some moment $x_{i+1} > x_{opt}(x_L^i)$ happens, after which $x_{opt}(x_L^i)$ moves monotonically "forward" to the right. This monotone property turns out to be quite useful to our algorithm. Similarly, we have Lemma 3 for S_R, which implies a monotone property of $x_{opt}(s_R^i)$. The proof of Lemma 3 is symmetric to that for Lemma 2, and we omit it.

Lemma 3. *For any two scenarios s_R^i and s_R^{i+1} of S_R with $1 \leq i \leq n-1$, if $x_{i+1} \leq x_{opt}(s_R^i)$, then $x_{i+1} \leq x_{opt}(s_R^i) \leq x_{opt}(s_R^{i+1})$; otherwise, $x_{opt}(s_R^{i+1}) \leq x_{opt}(s_R^i) \leq x_{i+1}$.*

Based on Lemmas 2 and 3, we present our algorithm for computing $x_{opt}(s)$ for all $s \in S$ as follows. We first compute $x_{opt}(s)$ for all $s \in S_L$, using Lemma 2.

We will compute $x_{opt}(s_L^i)$ in the index order $i = 1, 2, \ldots, n$. We assume we already have a data structure D that can compute the values $T_L(x, s)$ and $T_R(x, s)$ whenever needed for any x and $s \in S_L$. Initially, to determine $x_{opt}(s_L^1)$, we compute the values $T_L(x, s_L^1)$ and $T_R(x, s_L^1)$ for $x = x_1, x_2, \ldots$ in the (forward) order to find the smallest index i_1 such that $T_L(x_{i_1}, s_L^1) \geq T_R(x_{i_1}, s_L^1)$. As discussed in [10], $x_{opt}(s_L^1) \in [x_{i_1-1}, x_{i_1}]$ and can be determined in constant time. Next, we compute $x_{opt}(x_L^2)$. Assume $x_2 \leq x_{opt}(x_L^2)$. By Lemma 2, $x_2 \leq x_{opt}(x_L^2) \leq x_{opt}(x_L^1)$, we only need to search the portions of $T_L(x, s_L^2)$ and $T_R(x, s_L^2)$ for $x_2 \leq x \leq x_{opt}(s_L^1)$. To this end, we compute the values $T_L(x, s_L^2)$ and $T_R(x, s_L^2)$ by using D for $x = x_{i_1}, x_{i_1-1}, \ldots$ in the (backward) order to find the first index i_2 such that $T_L(x_{i_2}, s_L^2) \geq T_R(x_{i_2}, s_L^2)$ and $T_L(x_{i_2-1}, s_L^2) < T_R(x_{i_2-1}, s_L^2)$. As discussed in [10], $x_{opt}(s_L^2) \in [x_{i_2-1}, x_{i_2}]$ and can be determined in constant time.

In general, assume $x_{opt}(s_L^j)$ has been computed and $x_{j+1} \leq x_{opt}(s_L^j)$. Further, assume $x_{opt}(s_L^j)$ is known in the interval $[x_{i_j-1}, x_{i_j}]$. To compute $x_{opt}(s_L^{j+1})$, by Lemma 2, we have $x_{j+1} \leq x_{opt}(s_L^{j+1}) \leq x_{opt}(s_L^j)$. We compute the values $T_L(x, s_L^{j+1})$ and $T_R(x, s_L^{j+1})$ by D for $x = x_{i_j}, x_{i_j-1}, \ldots$ in the (backward) order to find the first index i_{j+1} such that $T_L(x_{i_{j+1}}, s_L^{j+1}) \geq T_R(x_{i_{j+1}}, s_L^{j+1})$ and $T_L(x_{i_{j+1}-1}, s_L^{j+1}) < T_R(x_{i_{j+1}-1}, s_L^{j+1})$. Again, $x_{opt}(s_L^{j+1}) \in [x_{i_{j+1}-1}, x_{i_{j+1}}]$ and can be determined in constant time.

We continue the same procedure until the first time we have computed $x_{opt}(s_L^k)$ with $x_{opt}(x_L^k) < x_{k+1}$ for an index k. We also have the interval $[x_{i_k-1}, x_{i_k}]$ that contains $x_{opt}(s_L^k)$. By Lemma 2, $x_{opt}(s_L^k) \leq x_{opt}(s_L^{k+1}) \leq x_{k+1}$. Hence, to compute $x_{opt}(s_L^{k+1})$, we need to search the portions of $T_L(x, s_L^{k+1})$ and $T_R(x, s_L^{k+1})$ for $x_{opt}(s_L^k) \leq x$. To this end, we compute the values $T_L(x, s_L^{k+1})$ and $T_R(x, s_L^{k+1})$ by D for $x = x_{i_k-1}, x_{i_k}, \ldots$ in the (forward) order to find the first index i_{k+1} such that

$T_L(x_{i_{k+1}}, s_L^{k+1}) \geq T_R(x_{i_{k+1}}, s_L^{k+1})$ and $T_L(x_{i_{k+1}-1}, s_L^{k+1}) < T_R(x_{i_{k+1}-1}, s_L^{k+1})$. Again, $x_{opt}(s_L^{k+1}) \in [x_{i_{k+1}-1}, x_{i_{k+1}}]$ and can be determined in constant time. Next, we compute $x_{opt}(s_L^{k+2})$. We have the following observation.

Observation 1. *$x_{opt}(s_L^{k+1}) < x_{k+2}$ holds.*

Proof. By Lemma 2, we have $x_{opt}(s_L^k) \leq x_{opt}(s_L^{k+1}) \leq x_{k+1}$. Due to $x_{k+1} < x_{k+2}$, the observation simply follows.

Due to the above observation, we can compute $x_{opt}(s_L^{k+2})$ in the similar way as $x_{opt}(s_L^{k+1})$. We continue this procedure to compute $x_{opt}(s_L^j)$ for $j = k+2, k+3, \ldots, n$. Note that similar observation as Observation 1 always holds (i.e., $x_{opt}(s_L^j) < x_{j+1}$ for any j with $k+1 \leq j \leq n-1$). The algorithm stops when $x_{opt}(s_L^n)$ is computed.

To analyze the running time, suppose any needed values $T_L(x, s)$ and $T_R(x, s)$ in the above algorithm can be computed in $O(T_D)$ time by using the data structure D; then we have the following lemma.

Lemma 4. *The values $x_{opt}(s)$ for all scenarios $s \in S_L$ can be computed in $O(n \cdot T_D)$ time.*

Proof. It is sufficient to show that the number of calls to D is $O(n)$ in the entire algorithm. We still use k to denote the smallest index with $x_{opt}(s_L^k) < x_{k+1}$. By the monotone property in Lemma 2, $x_{opt}(s_L^i)$ is moving monotonically to the left for $i = 1, 2, \ldots, k$, and $x_{opt}(s_L^i)$ is moving monotonically to the right for $i = k+1, k+2, \ldots, n$. When we compute $x_{opt}(s_L^i)$ for $i = 1, \ldots, k$, the x values for computing $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ are monotone decreasing. Therefore, when computing the values $x_{opt}(s_L^i)$'s for $i = 1, \ldots, k$, the total number of calls on D is $O(n)$. Analogously, when computing the values $x_{opt}(s_L^i)$'s for $i = k+1, \ldots, n$, the total number of calls on D is also $O(n)$. The lemma thus follows.

It remains to design the data structure D, which is given in Lemma 5. We should point out that our algorithm has a property that makes the design of our data structure in Lemma 5 easier. Specifically, in our algorithm for computing $x_{opt}(s)$ for all $s \in S_L$, when we are computing $x_{opt}(s_L^i)$, for any $1 \leq i \leq n$, we need to compute $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ for certain values of x. After $x_{opt}(s_L^i)$ is computed, we will never need to compute $T_L(x, s_L^i)$ and $T_R(x, s_L^i)$ for the scenario s_L^i again. Note that the corresponding algorithm in [10] does not have such a property. Due to the space limit, the proof of Lemma 5 is omitted.

Lemma 5. *In $O(n)$ time and $O(n)$ space, we can build a data structure D that can compute in $O(\log n)$ time (i.e., $T_D = O(\log n)$) any value $T_L(x, s)$ or $T_R(x, s)$ needed in our algorithm for computing $x_{opt}(s)$ for all $s \in S_L$.*

Combining Lemmas 4 and 5, the values $x_{opt}(s)$ for all scenarios $s \in S_L$ can be computed in $O(n \log n)$ time. Using the similar algorithm and Lemma 3, we can also compute the values $x_{opt}(s)$ for all scenarios $s \in S_R$ in $O(n \log n)$ time.

In summary, the values $x_{opt}(s)$ and $T(x_{opt}(s), s)$ for all scenarios $s \in S = S_L \cup S_R$ can be computed in $O(n \log n)$ time and $O(n)$ space.

4 Computing the Minmax Regret

Our goal is to find an *optimal location* x^* such that $R_{max}(x) = \max_{s \in \Sigma} R(x, s)$ is minimized at $x = x^*$, where $R(x, s) = T(x, s) - T(x_{opt}(s), s)$. Again, by Lemma 1, $R_{max}(x) = \max_{s \in S} R(x, s)$, which also implies that $R_{max}(x)$ is the upper envelope of the functions $R(x, s)$ for all $s \in S$.

Consider any scenario s. Since $T(x_{opt}(s), s)$ is a constant value and $T(x, s)$ is a convex function, $R(x, s)$ is also convex. Therefore, $R_{max}(x)$ is the upper envelope of a set of convex functions, which is also convex. To determine an optimal solution x^*, it is sufficient to determine the lowest point of the convex function $R_{max}(x)$. Due to the convex property of $R_{max}(x)$, we will use binary search to find its lowest point. Since $T(x, s)$ is the upper envelope of $T_L(x, s)$ and $T_R(x, s)$, $R_{max}(x)$ is the upper envelope of $T_L(x, s)$ and $T_R(x, s)$ for all $s \in S$.

The high-level of our algorithm for finding x^* is a binary search scheme on the values $x_1, x_2, \ldots, x_n$. For each value x_k considered in the binary search, we compute the value $R_{max}(x_k)$. To this end, we present an $O(n)$ time algorithm in Section 4.1 that can compute the values $T_L(x', s)$ and $T_R(x', s)$ for all $s \in S$, for any x', after which we can determine the value $R_{max}(x')$ in additional $O(n)$ time since we already know the values $T(x_{opt}, s)$ for all $s \in S$ in Section 3. Based on the function that gives the value $R_{max}(x_k)$, we can also determine which direction to do binary search in a standard way [14, 15]. The binary search will end up with either $x^* = x_i$ for some x_i or an interval (x_i, x_{i+1}) such that $x^* \in (x_i, x_{i+1})$. In the latter case, we finally determine x^* in additional $O(n)$ time by linear programming [14, 15] as follows. Note that for any scenario s, the value $T_L(x, s)$ for $x \in (x_i, x_{i+1})$ are given by the same function $f_L^j(x, s)$ for some j, and similar observation holds for $T_R(x, s)$. We find the functions giving the values in the interval (x_i, x_{i+1}) for $T_L(x, s_L^i)$, $T_R(x, s_L^i)$, $T_L(x, s_R^i)$, and $T_R(x, s_R^i)$, for $i = 1, \ldots, n$. This can be done in $O(n)$ time by the same algorithm in Section 4.1. Denote by F the $O(n)$ functions computed above. Hence, x^* is the x-coordinate of the lowest point p^* of the upper envelope of the functions in F. Note that every function of F defines a half-line that spans the interval (x_i, x_{i+1}). Hence, although each function of F is a half-line, p^* is also the lowest point of the upper envelope of the lines that contain the half-lines of F, and thus p^* can be computed in $O(n)$ time by linear programming [14, 15].

4.1 A Linear Time Algorithm for Computing $T(x', s)$ for all $s \in S$

In this section, we present an $O(n)$ time algorithm for computing $T(x', s)$ for all $s \in S$, for any x'. In other words, our goal is to compute the values $T_L(x', s_L^i)$, $T_R(x', s_L^i)$, $T_L(x', s_R^i)$, and $T_R(x', s_R^i)$, for $i = 1, \ldots, n$. We only discuss our algorithm for computing $T_L(x', s_L^i)$ for $i = 1, \ldots, n$ since the algorithms for the other three cases are quite similar. Further, for each $1 \leq i \leq n$, the function $f_L^j(x', s_L^i)$ that gives the value $T_L(x', s_L^i)$ is also determined by the algorithm.

For any $1 \leq i \leq j \leq n$, we define $\alpha(i, j) = \sum_{k=i}^{j}(w_k^+ - w_k^-)$. After $O(n)$ time preprocessing, given any i and j with $1 \leq i \leq j \leq n$, we can obtain the value

$\alpha(i,j)$ in constant time. We omit the preprocessing details and below we assume we have done the preprocessing. For convenience, we let $\alpha(i,j)=0$ if $i>j$.

Let x' be any value with $x_1 \le x' \le x_n$. We first determine the index i such that $x_{i-1} < x' \le x_i$. Thus, for any scenario s, only functions $f_L^t(x',s)$ with $1 \le t \le i-1$ are defined on $x=x'$, and any function $f_L^t(x,s)$ with $i \le t \le n$ does not define on $x=x'$. We compute the value $T_L(x,s_L^1)$, which can be done in $O(n)$ time, e.g., by computing $f_L^j(x',s_L^1)$ for each j with $1 \le j \le i-1$.

Let k be the index such that $T_L(x',s_L^1)$ is given by the function $f_L^k(x',s_L^1)$, e.g., $T_L(x',s_L^1)=f_L^k(x',s_L^1)$. Hence, $k \le i-1$. Lemma 6 will be useful later.

Lemma 6. *Consider a function $f_L^t(x,s)$ and a scenario s_L^j. If $1 \le t,j \le i-1$, then $f_L^t(x',s_L^j)=f_L^t(x',s_L^1)+\alpha(2,m)$ with $m=\min\{t,j\}$. This implies $f_L^t(x',s_L^t)=f_L^t(x',s_L^j)$ if $t \le j \le i-1$.*

Proof. Consider any t and j with $1 \le t,j \le i-1$. First of all, since $x_{i-1} < x' \le x_i$, $t \le i-1$, and $j \le i-1$, both functions $f_L^t(x,s_L^1)$ and $f_L^t(x,s_L^j)$ are defined on $x=x'$. Comparing with the scenario s_L^1, the weight of each vertex v_h for $2 \le h \le j$ increase by $w_h^+ - w_h^-$ in the scenario s_L^j, and the weights of all other vertices are the same as before. According to their definitions, we obtain that $f_L^t(x',s_L^1)=f_L^t(x',s_L^j)+\alpha(2,t)$ if $t \le j$, and $f_L^t(x',s_L^1)=f_L^t(x',s_L^j)+\alpha(2,j)$ if $t \ge j$. The lemma thus follows.

With the value $T_L(x',s_L^1)$, Lemma 7 computes $T_L(x',s_L^j)$ for $2 \le j \le k$.

Lemma 7. *If $k \ge 2$, for any scenario s_L^j with $2 \le j \le k$, $T_L(x',s_L^j)=T_L(x',s_L^1)+\alpha(2,j)$.*

Proof. Assume $k \ge 2$. Consider any scenario s_L^j with $2 \le j \le k$. We first prove a *claim* that $f_L^k(x',s_L^j) \ge f_L^t(x',s_L^j)$ for any $1 \le t \le i-1$.

Due to $T_L(x',s_L^1)=f_L^k(x',s_L^1)$, it holds that $f_L^k(x',s_L^1) \ge f_L^t(x',s_L^1)$ for any $1 \le t \le i-1$. Consider any t with $1 \le t \le i-1$. By Lemma 6, we have $f_L^t(x',s_L^j)=f_L^t(x',s_L^1)+\alpha(2,m)$, where $m=\min\{j,t\}$. Since $j \le k$, $f_L^k(x',s_L^j)=f_L^k(x',s_L^1)+\alpha(2,j)$ holds by Lemma 6. Clearly, $\alpha(2,j) \ge \alpha(2,m) \ge 0$ due to $m \le j$. Therefore, we obtain that $f_L^k(x',s_L^j) \ge f_L^t(x',s_L^j)$.

The above claim implies $T_L(x',s_L^j)=f_L^k(x',s_L^j)$. Since $f_L^k(x',s_L^j)=f_L^k(x',s_L^1)$ $+\,\alpha(2,j)$ by Lemma 6 and $T_L(x',s_L^1)=f_L^k(x',s_L^1)$, the lemma follows.

Suppose the value $T_L(x',s_L^{i-1})$ has already been computed; Lemma 8 shows how to obtain $T_L(x',s_L^j)$ for $i \le j \le n$. The proof of Lemma 8 is omitted.

Lemma 8. *For any scenario s_L^j with $i \le j \le n$, $T_L(x',s_L^j)=T_L(x',s_L^{i-1})$.*

Based on the preceding two lemmas, we can easily compute $T_L(x',s_L^j)$ for $j=2,\ldots,k$ in $O(n)$ time, and compute $T_L(x',s_L^j)$ for $j=i,\ldots,n$ in $O(n)$ time provided that we know the value $T_L(x',s_L^{i-1})$.

It remains to compute $T_L(x',s_L^t)$ for $t=k+1,\ldots,i-1$, for which we present an $O(n)$ time algorithm below. Note that our algorithm itself is simple, but it is not easy to discover the observations behind the scene. Our algorithm will compute a *solution index list* $K=\{k_1,k_2,\ldots,k_d\}$ with the following properties:

Property (1) $k_1 = k$ and $k_1 \leq k_2 \leq \cdots \leq k_d \leq i-1$.

Property (2) For any j with $1 \leq j \leq d-1$, $f_L^{k_j}(x', s_L^{k_j}) < f_L^{k_{j+1}}(x', s_L^{k_{j+1}})$ and $f_L^{k_j}(x', s_L^1) \geq f_L^{k_{j+1}}(x', s_L^1)$.

Property (3) For any j with $1 \leq j \leq d-1$, for any t with $k_j \leq t < k_{j+1}$, either $f_L^t(x', s_L^t) \leq f_L^{k_j}(x', s_L^{k_j})$ or $f_L^t(x', s_L^1) < f_L^{k_{j+1}}(x', s_L^1)$. If $k_d \neq i-1$, then for any t with $k_d \leq t \leq i-1$, $f_L^t(x', s_L^t) \leq f_L^{k_d}(x', s_L^{k_d})$.

If we already have such a solution index list K, Lemma 9 provides a way to compute the values $T_L(x', s_L^t)$ for $k+1 \leq t \leq i-1$ in $O(n)$ time. The proof of Lemma 9 is omitted.

Lemma 9. *For any scenario s_L^t with $k+1 \leq t \leq i-1$, if $k_j < t \leq k_{j+1}$ for some $1 \leq j \leq d-1$, then $T_L(x', s_L^t) = \max\{f_L^{k_j}(x', s_L^t), f_L^{k_{j+1}}(x', s_L^t)\}$; if $k_d \neq i-1$ and $k_d < t$, then $T_L(x', s_L^t) = f_L^{k_d}(x', s_L^t)$.*

Suppose we have a solution index list K. After we compute the values $f_L^{k_j}(x', s_L^1)$ for $j = 1, \ldots, d$ in $O(n)$ time, by Lemma 9 we can compute $T_L(x', s_L^t)$ for all $k+1 \leq t \leq i-1$ in $O(n)$ time (with the help of Lemma 6).

It remains to compute the solution index list K, for which we present a simple linear time algorithm as follows. We assume the values $f_L^t(x', s_L^1)$ for $t = 1, \ldots, i-1$ have been computed in $O(n)$ time.

Our algorithm will consider the indices incrementally from $t = k+1$ to $t = i-1$. A stack A is maintained during the algorithm to store a sequence of indices. Initially A contains only one index k, and after the algorithm finishes the index list in A from bottom to top is exactly K. For each t with $k+1 \leq t \leq i-1$, the algorithm proceeds as follows. Let k_j be the index on the top of the current stack A. We first compare the two values $f_L^t(x', s_L^1)$ and $f_L^{k_j}(x', s_L^1)$. Note that both values have been computed. If $f_L^t(x', s_L^1) > f_L^{k_j}(x', s_L^1)$, then we pop k_j out of A and consider the next top index on A (this is consistent with Property (3) of K and we ignore the detailed discussion on this). We claim that the stack A will never be empty because the index k is at the bottom of A. Indeed, recall that $T_L(x', s_L^1) = f_L^k(x', s_L^1)$ by the definition of k. Hence, $f_L^k(x', s_L^1) \geq f_L^m(x', s_L^1)$ for any $1 \leq m \leq i-1$, and in particular, $f_L^t(x', s_L^1) \leq f_L^k(x', s_L^1)$. Therefore, k will never be popped out of A. If $f_L^t(x', s_L^1) \leq f_L^{k_j}(x', s_L^1)$, we further compare the two values $f_L^t(x', s_L^t)$ and $f_L^{k_j}(x', s_L^{k_j})$. By Lemma 6, $f_L^t(x', s_L^t) = f_L^t(x', s_L^1) + \alpha(2, t)$ and $f_L^{k_j}(x', s_L^{k_j}) = f_L^{k_j}(x', s_L^1) + \alpha(2, k_j)$. Hence, both $f_L^t(x', s_L^t)$ and $f_L^{k_j}(x', s_L^{k_j})$ can be computed in constant time. If $f_L^t(x', s_L^t) > f_L^{k_j}(x', s_L^{k_j})$, then we push t on the top of A and set $k_{j+1} = t$ (this is consistent with Property (2) of K); otherwise, we ignore t (this is consistent with Property (3) of K) and proceed on $t+1$. After $t = i-1$ is considered, we terminate the algorithm and the index list in the stack A is our solution index list K. The running time of the algorithm is $O(n)$ because once an index is popped out of A it will never be considered again. We conclude this section with the following theorem.

Theorem 1. *The optimal position x^* for the minmax regret problem and the maximum regret $R_{max}(x^*)$ can be computed in $O(n \log n)$ time and $O(n)$ space.*

References

1. Averbakh, I., Bereg, S.: Facility location problems with uncertainty on the plane. Discrete Optimization 2, 3–34 (2005)
2. Averbakh, I., Berman, O.: Minimax regret p-center location on a network with demand uncertainty. Location Science 5, 247–254 (1997)
3. Averbakh, I., Berman, O.: Algorithms for the robust 1-center problem on a tree. European Journal of Operational Research 123, 292–302 (2000)
4. Averbakh, I., Berman, O.: Minmax regret median location on a network under uncertainty. INFORMS Journal on Computing 12, 104–110 (2000)
5. Averbakh, I., Berman, O.: An improved algorithm for the minmax regret median problem on a tree. Networks 2, 97–103 (2003)
6. Bhattacharya, B., Kameda, T.: A linear time algorithm for computing minmax regret 1-median on a tree. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 1–12. Springer, Heidelberg (2012)
7. Bhattacharya, B., Kameda, T., Song, Z.: Computing minmax regret 1-median on a tree network with positive/Negative vertex weights. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 588–597. Springer, Heidelberg (2012)
8. Bhattacharya, B., Kameda, T., Song, Z.: Minmax regret 1-center on a path/cycle/tree. In: Proc. of the Sixth International Conference on Advanced Engineering Computing and Applications in Sciences, pp. 108–113 (2012)
9. Chen, B., Lin, C.S.: Minmax-regret robust 1-median location on a tree. Networks 31, 93–103 (1998)
10. Cheng, S.-W., Higashikawa, Y., Katoh, N., Ni, G., Su, B., Xu, Y.: Minimax regret 1-sink location problems in dynamic path networks. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) TAMC 2013. LNCS, vol. 7876, pp. 121–132. Springer, Heidelberg (2013)
11. Conde, E.: A note on the minmax regret centdian location on trees. Operations Research Letters 36, 271–275 (2008)
12. Kamiyama, N., Katoh, N., Takizawa, A.: An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity. Transactions on Information and Systems E89-D (8), 2372–2379 (2006)
13. Kouvelis, P., Yu, G. (eds.): Robust discrete optimization and its applications. Kluwer Academic Publishers, Dordrecht (1997)
14. Megiddo, N.: Linear-time algorithms for linear programming in R^3 and related problems. SIAM Journal on Computing 12(4), 759–776 (1983)
15. Megiddo, N.: Linear programming in linear time when the dimension is fixed. Journal of the ACM 31(1), 114–127 (1984)
16. Puerto, J., Rodríguez-Chía, A., Tamir, A.: Minimax regret single-facility ordered median location problems on networks. INFORMS Journal on Computing 21, 77–87 (2009)
17. Yu, H.I., Lin, T.C., Wang, B.F.: Improved algorithms for the minmax-regret 1-center and 1-median problems. ACM Transactions on Algorithms 4(3), article No. 36 (2008)

Author Index