LNCS 10807

ARCoSS

Michael A. Bender
Martín Farach-Colton
Miguel A. Mosteiro (Eds.)

LATIN 2018: Theoretical Informatics

13th Latin American Symposium
Buenos Aires, Argentina, April 16–19, 2018
Proceedings

Springer

Michael A. Bender · Martín Farach-Colton
Miguel A. Mosteiro (Eds.)

LATIN 2018: Theoretical Informatics

13th Latin American Symposium
Buenos Aires, Argentina, April 16–19, 2018
Proceedings

Springer

Editors
Michael A. Bender
Stony Brook University
Stony Brook, NY
USA

Miguel A. Mosteiro
Pace University
New York, NY
USA

Martín Farach-Colton
Rutgers University
New Brunswick, NJ
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-77403-9 ISBN 978-3-319-77404-6 (eBook)
https://doi.org/10.1007/978-3-319-77404-6

Library of Congress Control Number: 2018937360

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains the papers presented at the 13th Latin American Theoretical Informatics Symposium (LATIN 2018) held during April 16–19, 2018, in Buenos Aires, Argentina. Previous editions of LATIN took place in São Paulo, Brazil (1992), Valparaíso, Chile (1995), Campinas, Brazil (1998), Punta del Este, Uruguay (2000), Cancún, México (2002), Buenos Aires, Argentina (2004), Valdivia, Chile (2006), Buzios, Brazil (2008), Oaxaca, México (2010), Arequipa, Perú (2012), Montevideo, Uruguay (2014), and Ensenada, México (2016).

The conference received 161 submissions from around the world. Each submission was reviewed by four Program Committee members, often with the help of additional external referees. After an electronic discussion, the committee selected 63 submissions for presentation.

The LATIN symposium featured keynote talks by Flavia Bonomo (Universidad de Buenos Aires), Leslie Goldberg (University of Oxford), Andrea Richa (Arizona State University), and Santosh Vempala (Georgia Institute of Technology).

The Program Committee was delighted to present the Alejandro Lopez-Ortiz Best Paper Award jointly to the following papers: "An Average-Case Lower Bound against ACC^0" by Ruiwen Chen, Igor Carboni Oliveira, and Rahul Santhanam, and "Time-Space Trade-Offs for Computing Euclidean Minimum Spanning Trees" by Bahareh Banyassady, Luis Barba, and Wolfgang Mulzer.

The Imre Simon Test-of-Time Award, which was started in 2012, is given to the authors of the LATIN paper deemed to be most influential among all those published at least ten years prior to the current edition of the conference. Papers published in the LATIN proceedings up to and including 2008 were eligible for the 2018 award. This year the winner was Marie-France Sagot for her paper "Spelling Approximate Repeated or Common Motifs Using a Suffix Tree," which appeared at LATIN 1998.

Many people helped to make LATIN 2018 possible. First, we would like to recognize the outstanding work of the members of the Program Committee. Their commitment contributed to a detailed discussion on each of the submitted papers. We also want to thank the Organizing Committee for their thorough job running LATIN 2018. Finally, we want to thank the Steering Committee for their advice and feedback.

We are grateful for the facilities provided by EasyChair for paper evaluation and the preparation of this volume.

April 2018

Michael A. Bender
Martín Farach-Colton
Miguel A. Mosteiro

The Imre Simon Test-of-Time Award

The LATIN 2018 winner of the Imre Simon Test-of-Time Paper Award considering papers up to the 2008 edition of the conference is:

Spelling Approximate Repeated or Common Motifs Using a Suffix Tree, by Marie-France Sagot, LATIN 1998, LNCS 1380, 374–390, 1998.

Sequence motifs are recurring patterns (arrangement of characters) that may have some biological significance for example in DNA studies. When the sequence of characters of the motif is known, search methods can take advantage of such information to find the motif in the sequence. Classical algorithms for this task include Boyer-Moore, Rabin-Karp, suffix trees, and so on. Central problems are then the discovery of motifs, that is to find the order in which their characters appear, and the study of repetitions of motifs from a sequence.

In her LATIN'98 paper, Sagot gives algorithmic solutions to the latter problem when up to a certain number of mismatches are allowed and the motif appears a given minimum number of times. The algorithms consider that the motif may not be exactly present in the sequence. These motifs are called models. Waterman introduced this type of studies in the 1980's. Improved algorithms were given by several authors including Baeza-Yates and Myers, among others. The algorithms in Sagot's LATIN'98 article are not only elegant but also efficient by avoiding costly traversals of the suffix tree. Moreover, by pruning branches in that tree when constraints conditions are not satisfied, the complexity is further reduced. Sagot's article is seminal in the sense that it is the first systematic use of suffix trees for the purpose of motif extraction.

In considering Sagot's paper for the award, the selection committee was impressed by the relevance of the problem addressed, originality of the technique used to solve it, clarity of presentation, widespread recognition, and number of citations in the literature.

Marcos Kiwi
Daniel Panario
Jacques Sakarovitch

Organization

Program Committee

Eric Allender	Rutgers University, USA
Gabriela Araujo-Pardo	Universidad Nacional Autónoma de México, Mexico
Esther Arkin	Stony Brook University, USA
Jérémy Barbay	Universidad de Chile, Chile
Michael A. Bender	Stony Brook University, USA
Vladimir Braverman	Johns Hopkins University, USA
Luciana Buriol	Universidade Federal do Rio Grande do Sul, Brazil
Armando Castañeda	Universidad Nacional Autónoma de México, Mexico
Keren Censor-Hillel	Technion Israel Institute of Technology, Israel
Witold Charatonik	University of Wroclaw, Poland
Jing Chen	Stony Brook University, USA
Giorgos Christodoulou	University of Liverpool, UK
Guy Even	Tel Aviv University, Israel
Cristina G. Fernandes	University of São Paulo, Brazil
Antonio Fernández Anta	IMDEA Networks Institute, Spain
Paolo Ferragina	University of Pisa, Italy
Celina De Figueiredo	Universidade Federal do Rio de Janeiro, Brazil
Jeremy Fineman	Georgetown University, USA
Johannes Fischer	Technische Universität Dortmund, Germany
Paola Flocchini	University of Ottawa, Canada
Lance Fortnow	Georgia Institute of Technology, USA
Pierre Fraigniaud	CNRS and University of Paris Diderot, France
Juan Garay	Texas A&M University, USA
Leszek Gasieniec	University of Liverpool, UK
Seth Gilbert	National University of Singapore
Julián Gutierrez	University of Oxford, UK
Inge Li Gørtz	Technical University of Denmark
John Iacono	New York University, USA
Taisuke Izumi	Nagoya Institute of Technology, Japan
Jesper Jansson	The Hong Kong Polytechnic University, SAR China
Gabriela Jerónimo	Universidad de Buenos Aires, Argentina
Artur Jeiż	University of Wroclaw, Poland
Rob Johnson	VMware Research
Tomasz Jurdzinski	University of Wroclaw, Poland
Shuji Kijima	Kyushu University, Japan
Michal Koucky	Charles University, Czech Republic
Yiannis Koutis	University of Puerto Rico
Sławomir Lasota	University of Warsaw, Poland
Reut Levi	Max-Planck-Institut für Informatik, Germany

Min Chih Lin	Universidad de Buenos Aires, Argentina
Claudia Linhares Sales	Universidade Federal do Ceara, Brazil
Javier Marenco	Universidad Nacional de General Sarmiento and Universidad de Buenos Aires, Argentina
Conrado Martínez	Universitat Politècnica de Catalunya, Spain
Moti Medina	Max-Planck-Institut für Informatik, Germany
Joseph S. B. Mitchell	Stony Brook University, USA
Marco Molinaro	Pontifícia Universidade Católica do Rio de Janeiro, Brazil
Miguel A. Mosteiro	Pace University, USA
Marcelo Mydlarz	Universidad Nacional de General Sarmiento and CONICET, Argentina
Calvin Newport	Georgetown University, USA
Igor Potapov	University of Liverpool, UK
Jared Saia	University of New Mexico, USA
Rodrigo Silveira	Universitat Politècnica de Catalunya, Spain
José A. Soto	Universidad de Chile, Chile
Paul Spirakis	University of Liverpool and University of Patras, UK/Greece
Grzegorz Stachowiak	University of Wroclaw, Poland
Maya Stein	Universidad de Chile, Chile
Frank Stephan	National University of Singapore
Christopher Thraves	Universidad de Concepción, Chile
Denis Trystram	Grenoble Alpes University, France
José Verschae	Pontificia Universidad Católica de Chile, Chile
Mark Daniel Ward	Purdue University, USA
Andreas Wiese	Universidad de Chile, Chile
Prudence Wong	University of Liverpool, UK
Yukiko Yamauchi	Kyushu University, Japan
Maxwell Young	Mississippi State University, USA

Additional Reviewers

Abrahamsen, Mikkel
Afshani, Peyman
Aggarwal, Abhinav
Akrida, Eleni C.
Alcantara, Manuel
Almeida, Sheila
Amir, Amihood
Andoni, Alexandr
Araujo, Julio
Arias, Jaime
Azar, Yossi
Bahamondes, Bastián
Balko, Martin
Barmak, Jonathan
Bartal, Yair
Baste, Julien
Bell, Paul
Benevides, Fabricio S.
Berenbrink, Petra
Berndt, Sebastian
Bille, Philip
Blaum, Manuela
Bodlaender, Hans L.
Boettcher, Stefan
Bornstein, Claudson
Bougeret, Marin
Bourhis, Pierre
Bournez, Olivier
Braga, Mónica
Braga, Rodrigo
Bringmann, Karl
Buchin, Maike
Byrka, Jaroslaw
Béal, Marie-Pierre
Böhm, Martin
Ceccarello, Matteo
Chabchoub, Yousra
Chalermsook, Parinya
Cheong, Otfried

Chitnis, Rajesh
Chiu, Hua-Sheng
Chiu, Kenny
Coelho de Pina, Jose
Coelho, Rafael
Courcelle, Bruno
Cournier, Alain
Cseh, Ágnes
Cung, Van Dat
Cunha, Luis
Czyzowicz, Jurek
Da Fonseca,
Guilherme Dias
Dabrowski, Konrad
Dabrowski, Konrad
Kazimierz
Daescu, Ovidiu
Das, Bireswar
Datta, Ajoy K.
de Lima, Paloma
De Loera, Jesús
de Oliveira Oliveira,
Mateus
de Rezende, Pedro J.
Debiasio, Louis
Deligkas, Argyrios
Delle Donne, Diego
Dominik, Köppl
Dory, Michal
Driemel, Anne
Dudek, Bartek
Dudycz, Szymon
Durocher, Stephane
Dvorak, Pavel
Dvorak, Zdenek
Dybdahl Ahle, Thomas
Dürr, Christoph
Edwards, Katherine
Epstein, Leah
Escalante, Mariana
Escoffier, Bruno
Eto, Hiroshi
Ettienne, Mikko Berggren
Fabila-Monroy, Ruy
Fagerberg, Rolf
Faure, Adrien
Fernau, Henning
Feuerstein, Esteban
Fichtenberger, Hendrik
Fijalkow, Nathanaël
Fluschnik, Till
Freire, Alexandre
Frieze, Alan
Fulek, Radoslav
García Pardo, Eduardo
Garncarek, Pawel
Gaspers, Serge
Gawrychowski, Pawel
Gańczorz, Michał
Ghosh, Shamik
Giannopoulos, Panos
Gliesch, Alex
Goddard, Wayne
Gonze, François
Gonçalves, Daniel
Goranci, Gramoz
Gorecki, Pawel
Guedes, André L. P.
Gupta, Diksha
Gusev, Vladimir
Hagerup, Torben
Har-Peled, Sariel
Herrero, María Isabel
Heydrich, Sandy
Hirvensalo, Mika
Horiyama, Takashi
Huemer, Clemens
Ibarra, Louis
Ibsen-Jensen, Rasmus
Italiano, Giuseppe F.
Jeż, Łukasz
Jiménez, Andrea
Kanté, Mamadou
Moustapha
Keil, Mark
Kesselheim, Thomas
Khoury, Seri
Kiwi, Marcos
Kiyomi, Masashi
Kleiman, Elena
Klein, Rolf
Klimm, Max
Knauer, Kolja
Knop, Dušan
Ko, Sang-Ki
Koch, Ivo
Koivisto, Mikko
Korman, Matias
Kortsarz, Guy
Kostitsyna, Irina
Kowalik, Lukasz
Kraska, Artur
Kratsch, Stefan
Krumpe, Filip
Krysta, Piotr
Kunysz, Adam
Kurpicz, Florian
Kynčl, Jan
Kézdy, André
Łacki, Jakub
Lampis, Michael
Lamprou, Ioannis
Lang, Harry
Laurent, Monique
Lenzen, Christoph
Levcopoulos, Christos
Levin, Keith
Li, Bo
Liu, Hsiang-Hsuan
Lopes, Fabio
Lucarelli, Giorgio
Madduri, Kamesh
Maia, Ana Karolinna
Mallmann-Trenn, Frederik
Marcinkowski, Jan
Marino, Andrea
Martin, Russell
Matera, Guillermo
Mayer, Tyler
Mazzoleni, María Pía
McCauley, Samuel
Melissourgos,
Themistoklis
Menezes, Alfred
Mertzios, George
Mezzini, Mauro
Mikulski, Lukasz
Mizrahi, Michel J.

Molloy, Michael
Montealegre, Pedro
Mouawad, Amer
Moura, Phablo
Mozes, Shay
Mulzer, Wolfgang
Musial, Jedrzej
Mydlarz, Marcelo
Najib, Muhammad
Nederlof, Jesper
Nenadov, Rajko
Nichterlein, André
Niklitschek, Sebastian
Niskanen, Reino
Nogueira, Loana
Nowicki, Krzysztof
Obdrzalek, Jan
Oh, Eunjin
Olarte, Carlos
Oliveira, Fabiano
Oliveros, Deborah
Ooshita, Fukuhito
Otop, Jan
Pagh, Rasmus
Paixao, Joao
Palfrader, Peter
Paluch, Katarzyna
Panagopoulou, Panagiota
Panolan, Fahad
Parys, Paweł
Paz, Ami
Pedrosa, Lehilton L. C.
Pelayo, Ignacio M.
Pemmaraju, Sriram
Penso, Lucia Draque
Pereira, André Grahl
Perifel, Sylvain
Perrucci, Daniel
Phillips, Cindy
Posner, Daniel
Possani, Edgar
Préa, Pascal
Psarros, Ioannis
Pérez-Lantero, Pablo
Quesada, Luis
Rabinovich, Yuri
Rajasekaran, Senthil
Raptopoulos, Christoforos
Rau, Malin
Ravi, R.
Razenshteyn, Ilya
Rojas, Javiel
Rosone, Giovanna
Rotenberg, Eva
Roy, Bodhayan
Rubio-Montiel, Christian
Rué, Juanjo
Sabia, Juan
Safe, Martín Darío
Salamon, Andras
Samal, Robert
Sampaio, Rudini
San Felice, Mário César
Sankowski, Piotr
Satti, Srinivasa Rao
Schewior, Kevin
Schlachter, Uli
Schmidt, Melanie
Schouery, Rafael
Schwiegelshohn, Chris
Seara, Carlos
Seco, Diego
Semikhin, Pavel
Serna, Maria
Servedio, Rocco
Shallit, Jeffrey
Shalom, Mordo
Sheehy, Don
Shi, Yangguang
Shiraga, Takeharu
Shrestha, Yash Raj
Shun, Julian
Silva, Ana
Sinclair, Alistair
Singh, Shikha
Soares, Ronan
Soulignac, Francisco
Souza, Críston
Souza, Ueverton
Srivastav, Abhinav
Sucupira, Rubens
Sudo, Yuichi
Szegedy, Mario
Szykuła, Marek
Telha, Claudio
Thaler, Justin
Totzke, Patrick
Trehan, Amitabh
Ueckerdt, Torsten
Uitto, Jara
Uno, Yushi
Upadhyay, Jalaj
Vahrenhold, Jan
Valencia-Pabon, Mario
van Stee, Rob
Vassiliev, Saveli
Verdugo, Victor
Viglietta, Giovanni
Vigneron, Antoine
Viola, Emanuele
Wagner, Frederic
Walczak, Bartosz
Wang, Haitao
Ward, Justin
Wasa, Kunihiro
Wei, Fan
Wilfong, Gordon
Winkler, Peter
Winslow, Andrew
Witkowski, Piotr
Woodall, William
Wright, John
Wulff-Nilsen, Christian
Yang, Lin
Zamora, José
Zehavi, Meirav
Zhang, Peng
Zhou, Samson
Zito, Michele
Zokaei Ashtiani, Hassan

Contents

The Graph Tessellation Cover Number: Extremal Bounds, Efficient Algorithms and Hardness

Alexandre Abreu[1], Luís Cunha[2], Tharso Fernandes[3,4], Celina de Figueiredo[1](✉), Luis Kowada[2], Franklin Marquezino[1], Daniel Posner[1], and Renato Portugal[4]

[1] Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil
{santiago,celina,franklin,posner}@cos.ufrj.br
[2] Universidade Federal Fluminense, Niterói, Brazil
{lfignacio,luis}@ic.uff.br
[3] Universidade Federal do Espírito Santo, Vitória, Brazil
[4] Laboratório Nacional de Computação Científica, Petrópolis, Brazil
{tharsodf,portugal}@lncc.br

Abstract. A tessellation of a graph is a partition of its vertices into vertex disjoint cliques. A tessellation cover of a graph is a set of tessellations that covers all of its edges. The t-TESSELLABILITY problem aims to decide whether there is a tessellation cover of the graph with t tessellations. This problem is motivated by its applications to quantum walk models, in especial, the evolution operator of the staggered model is obtained from a graph tessellation cover. We establish upper bounds on the tessellation cover number given by the minimum between the chromatic index of the graph and the chromatic number of its clique graph and we show graph classes for which these bounds are tight. We prove $\mathcal{NP}$-completeness for t-TESSELLABILITY if the instance is restricted to planar graphs, chordal $(2,1)$-graphs, $(1,2)$-graphs, diamond-free graphs with diameter five, or for any fixed t at least 3. On the other hand, we improve the complexity for 2-TESSELLABILITY to a linear-time algorithm.

Keywords: Staggered quantum walk · Clique graph · Tessellation

1 Introduction

Random walks play an important role in Computer Science mainly in the area of algorithms and it is expected that quantum walks, which is the quantum counterpart of random walks, will play at least a similar role in Quantum Computation. In fact, the interest in quantum walks has grown considerably in the last decades, especially because they can be used to build quantum algorithms that outperform their classical counterparts [1].

This work was partially supported by the Brazilian agencies CAPES, CNPq and FAPERJ.

Recently, the staggered quantum walk model [2] was proposed. This model is defined by an evolution operator, which is described by a product of local unitary matrices obtained from a *graph tessellation cover*. A tessellation is a partition of the vertices of a graph into vertex disjoint cliques, and a tessellation cover is a set of tessellations so that the union covers the edge set. In order to fully understand the possibilities of the staggered model, it is fundamental to introduce the t-TESSELLABILITY problem. This problem aims to decide whether a given graph can be covered by t tessellations.

The simplest evolution operators are the product of few local unitary matrices, and at least two matrices (corresponding to 2-tessellable graphs) are required. There is a recipe to build a local unitary matrix based on a tessellation [2]. Each clique of the partition establishes a neighborhood around which the walker can move under the action of the local unitary matrix. To define the evolution operator of the quantum walk, one has to include extra tessellations until the tessellation union covers the edge set. Figure 1 depicts an example of how a quantum walker could spread across the vertices of a graph, given a particular tessellation cover. Bold vertices represent non-zero amplitudes, meaning that a measurement of the position can reveal the walker at one of those vertices. Note that after each step the walker spreads across the cliques in the corresponding tessellation.

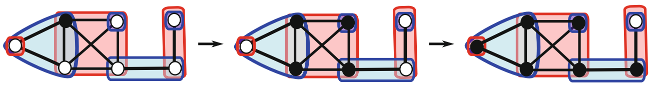


Fig. 1. The spreading of a walker subject to locality across a 2-tessellable graph. At each step, the walker may be observed at bold vertices.

The study of tessellations in the context of Quantum Computing was proposed by Portugal et al. [2] with the goal of obtaining the dynamics of quantum walks. Portugal analyzed the 2-tessellable case in [3] and we described examples for the t-tessellable case in [4]. The present work is the first systematic attempt to study the tessellation problem as a branch of Graph Theory. Our aim is the study of graph classes with extremal tessellation cover numbers, efficient algorithms, and hardness.

We describe upper bounds in Sect. 2. We establish graph classes for which these bounds are tight in Sect. 3. We use these graphs with extremal tessellation covers to establish hardness results for several graph classes in Sect. 4. We obtain proofs of t-TESSELLABILITY $\mathcal{NP}$-completeness for planar graphs, chordal $(2, 1)$-graphs, $(1, 2)$-graphs, diamond-free graphs with diameter five, or if t is fixed for $t \geq 3$. Moreover, we describe a linear-time algorithm for 2-TESSELLABILITY.

2 Preliminaries on the Tessellation Cover Number

A *clique* is a subset of vertices of a graph such that its induced subgraph is complete. The size of a maximum clique of a graph G is denoted by $\omega(G)$.

The *clique graph* $K(G)$ is the intersection graph of the maximal cliques of G. A *partition of the vertices of a graph into cliques* is a collection of vertex disjoint cliques, where the union of these cliques is the vertex set. Clique graphs play a central role in tessellation covers. (See [5] for an extensive survey on clique graphs and [6] for omitted graph theory terminologies).

Definition 1. A *tessellation* $\mathcal{T}$ is a partition of the vertices of a graph into cliques. An edge *belongs* to the tessellation $\mathcal{T}$ if and only if its endpoints belong to the same clique in $\mathcal{T}$. The set of edges belonging to $\mathcal{T}$ is denoted by $\mathcal{E}(\mathcal{T})$.

Definition 2. Given a graph G with edge set $E(G)$, a *tessellation cover* of size t of G is a set of t tessellations $\mathcal{T}_1, \ldots, \mathcal{T}_t$, whose union $\cup_{i=1}^{t} \mathcal{E}(\mathcal{T}_i) = E(G)$. A graph G is called *t-tessellable* if there is a tessellation cover of size at most t. The t-TESSELLABILITY PROBLEM aims to decide whether a graph G is t-tessellable. The *tessellation cover number* $T(G)$ is the size of a smallest tessellation cover of G.

A *coloring* (resp. an *edge-coloring*) of a graph is a labeling of the vertices (resp. edges) with colors such that no two adjacent vertices (resp. incident edges) have the same color. A *k-colorable* (resp. *k-edge-colorable*) graph is the one which admits a *coloring* (resp. an *edge-coloring*) with at most k colors. The *chromatic number* $\chi(G)$ (resp. *chromatic index* $\chi'(G)$) of a graph G is the smallest number of colors needed to color the vertices (resp. edges) of G.

Note that an edge-coloring of a graph G induces a tessellation cover of G. Each color class induces a partition of the vertex set into disjoint cliques of size two (vertices incident to edges of that color) and cliques of size one (vertices not incident to edges of that color), which forms a tessellation. Moreover, a coloring of $K(G)$ induces a tessellation cover of G. As presented in [4], two vertices of the same color in $K(G)$ correspond to disjoint maximal cliques of G and every edge of G is in at least one maximal clique. So, each color in $K(G)$ defines a tessellation in G by possibly adding cliques of size one (vertices that do not belong to maximal cliques of G, which are related to vertices of $K(G)$ with that color), such that the union of these tessellations is the edge set of G. Hence, we have the following upper bounds.

Theorem 1. *If G is a graph, then $T(G) \leq \min\{\chi'(G), \chi(K(G))\}$.*

Portugal [3] characterized the 2-tessellable graphs as those whose clique graphs are bipartite graphs. In order to characterize t-tessellable graphs, for $t \geq 3$, we were able to find graph classes such that $T(G) = 3$, with $\chi'(G)$ and $\chi(K(G))$ arbitrarily large, and additionally graph classes with extremal values of Theorem 1, i.e. $T(G) = \chi'(G)$ but $\chi(K(G))$ arbitrarily large; and $T(G) = \chi(K(G))$ but $\chi'(G)$ arbitrarily large, some of those examples were described in [4].

An interesting case occurs for a triangle-free graph. Note that any of its tessellations can only be formed by cliques of size two or one. Hence, we have an extremal result that if G is a triangle-free graph, then $T(G) = \chi'(G)$.

A graph is (k, ℓ) if its vertex set can be partitioned into k stable sets and ℓ cliques. Particularly, $(2, 0)$-graphs are *bipartite graphs*. It is known how to Δ-edge-color bipartite graphs [7] and to Δ-edge-color $\{\text{triangle}, \text{proper major}\}$-free graphs [8] in polynomial time. Therefore, not only t-TESSELLABILITY is polynomial time solvable for bipartite graphs and for $\{\text{triangle}, \text{proper major}\}$-free graphs, but there are also polynomial time procedures to obtain a minimum tessellation cover for these graph classes. Besides that, it is known that 3-EDGE COLORABILITY of triangle-free graphs is $\mathcal{NP}$-complete [9]. Therefore, 3-TESSELLABILITY of triangle-free graphs is also $\mathcal{NP}$-complete.

3 Extremal Tessellation Covers

Throughout the paper, an extremal tessellation cover of a graph is one of its tessellation covers whose size reaches an upper bound of Theorem 1. We are particularly interested in constructing graphs for which these extremal tessellation covers correspond (or are close) to an optimal edge coloring or an optimal vertex coloring of the clique graph of the original graph. For the sake of convenience, we may omit one-vertex cliques inside tessellations in our proofs.

Construction 1. Let H be obtained from a graph G by adding a star with $\chi'(G)$ leaves and identifying one of these leaves with a minimum degree vertex of G.

The tessellation cover number of H, obtained from Construction 1 on a non-regular graph G, is equal to its original chromatic index, i.e., $T(H) = \chi'(H) = \chi'(G)$. For regular graphs, if $\chi'(G) = \Delta + 1$, then $T(H) = \chi'(H) = \chi'(G)$. Otherwise, $T(H) = \chi'(H) = \chi'(G) + 1$. Construction 1 also implies that every non-regular graph G is subgraph of a graph H with $T(H) = \chi'(H) = \chi'(G)$.

Similarly, there exists a construction in diamond-free graphs, which are the $\{K_4 \backslash \{e\}\}$-free graphs, to force the tessellation cover number to be equal to the original chromatic number of the clique graph. First, we define a property of the cliques on a tessellation called exposed maximal clique. Such a property helps us with particular cases of diamond-free graphs.

Definition 3. A maximal clique K of a graph G is said *exposed* by a tessellation cover $\mathcal{C}$ if $E(K) \not\subseteq \mathcal{E}(\mathcal{T})$ for all $\mathcal{T} \in \mathcal{C}$, that is, the edges of K are covered by no tessellation of $\mathcal{C}$.

Lemma 1. *A graph G admits a minimum tessellation cover with no exposed maximal cliques if and only if $T(G) = \chi(K(G))$.*

In the remaining part of this section we consider diamond-free graphs, which have the following properties [10]: (1) their clique-graphs are diamond-free, and (2) any two maximal cliques intersect in at most one vertex.

Theorem 2. *If G is a diamond-free graph with $\chi(K(G)) = \omega(K(G))$, then $T(G) = \chi(K(G))$.*

A graph is K-*perfect* if its clique graph is perfect [11]. Since a diamond-free K-perfect graph G satisfies the premises of Theorem 2, we have $T(G) = \chi(K(G))$. Note that the size of the clique graph of a diamond-free graph is polynomially bounded by the size of the original graph. Moreover, there is a polynomial-time algorithm to obtain an optimal coloring of $K(G)$ with $\omega(K(G))$ colors [12] and, by Theorem 1, a coloring of $K(G)$ with t colors yields that G is t-tessellable. Thus, both the tessellation cover number and a minimum tessellation cover of diamond-free K-perfect graphs are obtained in polynomial time.

Interestingly, there are diamond-free graphs whose clique graphs have chromatic number greater than the tessellation cover number. Figure 2 illustrates an example of a 3-tessellable diamond-free graph whose clique graph has chromatic number 4 (the clique graph $K(G)$ is the Grötzsch graph, i.e. Mycielskian of a 5-cycle graph). Note that any minimum tessellation cover of this graph necessarily has an exposed maximal clique. Moreover, this graph shows that the upper bound of Theorem 3 is tight.

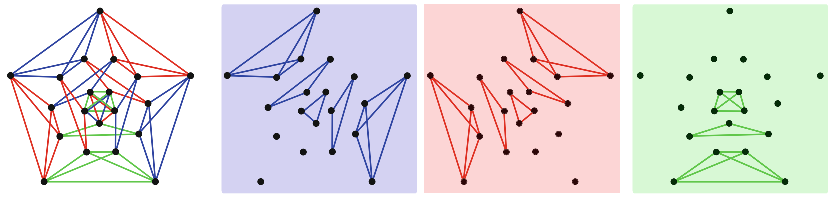


Fig. 2. Example of a 3-tessellable graph G whose clique graph is the Mycielskian of a C_5, with $\chi(K(G)) = 4$ but $T(G) = 3$. Each tessellation is depicted separately.

Lemma 2. *Let G be a 3-tessellable diamond-free graph. If C_1 and C_2 are two maximal cliques of G with a common vertex, then C_1 and C_2 cannot be both exposed by a minimum tessellation cover.*

Theorem 3. *If G is a 3-tessellable diamond-free graph, then $3 \leq \chi(K(G)) \leq 4$.*

We finish this section with a construction which forces the tessellation cover number of a graph H, obtained from Construction 2 on a diamond-free graph G, to be $T(H) = \chi(K(H)) = \chi(K(G))$. If G has $T(G) < \chi(K(G))$, then there is no vertex that belongs to $\chi(K(G))$ maximal cliques. The graph H obtained from G by Construction 2 satisfies $\chi(K(H)) = \chi(K(G))$ and contains a vertex that belongs to $\chi(K(G))$ maximal cliques, which implies $T(H) = \chi(K(G))$.

Construction 2. Let H be obtained from a graph G by iteratively adding pendant vertices to a vertex of G until it belongs to $\chi(K(G))$ maximal cliques.

Construction 2 implies that every diamond-free graph G is a subgraph of a graph H with $T(H) = \chi(K(H)) = \chi(K(G))$. Note that this construction is not restricted to diamond-free graphs and it can also be applied several times to vertices that only belong to one maximal clique. The hardness proofs of Theorems 5 and 6 rely on this result.

4 Computational Complexity

Now, we focus on the computational complexity of t-TESSELLABILITY, by firstly proving that the problem is in $\mathcal{NP}$. In Sect. 4.1, we use extremal tessellation covers obtained in the previous section to show $\mathcal{NP}$-completeness for any fixed $t \geq 3$ and when the problem instance is restricted to some graph classes. In Sect. 4.2, we efficiently solve 2-TESSELLABILITY in linear time.

Lemma 3. *t-TESSELLABILITY is in $\mathcal{NP}$.*

Proof. Consider a certificate for an instance of t-TESSELLABILITY, which consists of at most t tessellations of a given graph G. Note that each tessellation has at most m edges. Moreover, by Theorem 1 and the well-known Vizing's theorem on edge-colorability, if $t \geq \Delta+1$, then the answer is automatically *YES*. Otherwise, one can easily verify in polynomial time if the at most m edges in each of the at most $t \leq \Delta + 1$ tessellations form disjoint cliques in G and if the at most $m(\Delta + 1)$ edges in these tessellations cover all edges of G. □

4.1 $\mathcal{NP}$-completeness

A graph is *planar* if it can be embedded in the plane such that no two edges cross each other. We show a polynomial transformation from the $\mathcal{NP}$-complete 3-COLORABILITY of planar graphs with maximum degree four [7] to 3-TESSELLABILITY of planar graphs with maximum degree six.

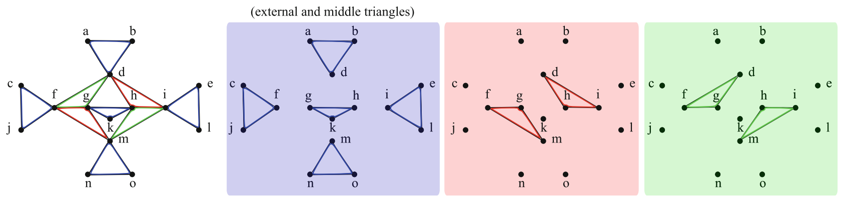


Fig. 3. The 3-tessellable *graph-gadget* of Lemma 4. Each tessellation is depicted separately. The external vertices are a, b, c, e, j, l, n, o, and the internal vertices are the remaining ones.

Lemma 4. *Any tessellation cover of size 3 of the* graph-gadget *depicted in Fig. 3 contains a tessellation that has the middle and the external triangles.*

Construction 3. Let graph H be obtained from a graph G by local replacements of the vertices of G such that each vertex u of G represents a graph-gadget of Fig. 3 denoted by u-gadget and each edge uv of G represents the intersection of the u-gadget with the v-gadget by identifying two external vertices of external triangles of those graph-gadgets.

Theorem 4. 3-TESSELLABILITY *of planar graphs with $\Delta \leq 6$ is $\mathcal{NP}$-complete.*

Proof. Let G be an instance graph of 3-COLORABILITY of planar graphs with $\Delta \leq 4$ and H be obtained by Construction 3 on G. Notice that applying Construction 3 on a planar graphs with $\Delta \leq 4$ results on a planar graph with $\Delta \leq 6$.

Suppose that G is 3-colorable. Then, H is 3-tessellable because the middle and the external triangles of a v-gadget can be covered by the tessellation related to the color of v and the remaining triangles of the v-gadget can be covered by the other two tessellations.

Suppose that H is 3-tessellable. Then, G is 3-colorable because the color of v in G can be related to the tessellation that covers the middle triangle of the v-gadget. This assignment is a 3-coloring because by Lemma 4 all external triangles of the v-gadget belong to the same tessellation of the middle triangle. The external triangles of the v-gadget are connected to the external triangles of the graph-gadgets of the neighborhood of v. Then, the tessellations of the latter external triangles must differ from the external triangles of the v-gadget. This implies that the neighborhood of vertex v receives different colors from the color of v. $\square$

The next construction allows us to show a hardness proof of t-TESSELLABILITY for any fixed $t \geq 4$.

Construction 4. Let H be a graph obtained from a graph G and a subset $F \subseteq V(G)$ as follows. Initially H is equal to G. Let $F = \{v_1, \ldots, v_{|F|}\}$ be a subset of vertices of $V(H)$. Add to H a complete graph $U = \{u_1, \ldots, u_{|F|}\}$. Add three vertices c_1, c_2 and c_3 adjacent to all vertices of U. Consider an integer $t \geq 4$. For each c_i $(1 \leq i \leq 3)$, add $t-1$ pendant vertices incident to c_i. For each $1 \leq j \leq |F|$, add an edge $v_j u_j$ and vertices $w_{j,l}$ for $1 \leq l \leq t-3$ adjacent to both v_j and u_j. For each vertex $w_{j,l}$, add $t-1$ pendant vertices incident to $w_{j,l}$.

Theorem 5. t-TESSELLABILITY *for any fixed $t \geq 4$ is $\mathcal{NP}$-complete.*

Sketch of the proof. Let G be an instance graph of 3-COLORABILITY of planar graphs with $\Delta \leq 4$. Let H' be the graph obtained from Construction 3 on G. Let H be the graph obtained from Construction 4 on H' with F being the set of all internal vertices of all graph-gadgets of H'.

Let $H[H']$ be the induced subgraph of H by the vertices of H'. H is t-tessellable (for $t \geq 4$) if and only $H[H']$ is 3-tessellable and the $\mathcal{NP}$-completeness follows immediately from Theorem 4. $\square$

Next, we show a polynomial transformation from the $\mathcal{NP}$-complete 3-COLORABILITY [7] to 4-TESSELLABILITY of chordal $(2,1)$-graphs. This proof is based on a result of Bodlaender et al. [13] for 3-$L(0,1)$-COLORING of split graphs.

Construction 5. Let H be a graph obtained from a non-bipartite graph G as follows. Initially $V(H) = V(G) \cup E(G)$ and $E(H) = \emptyset$. Add edges to H so that the $E(G)$ vertices induce a clique. For each $e = vw \in E(G)$, add to H edges ve and we. For each vertex $v \in V(H) \cap V(G)$, add three pendant vertices incident to v. Add a vertex u adjacent to all $E(G)$ vertices. Add three pendant vertices incident to u. Denote all pendant vertices by V_2.

Theorem 6. 4-TESSELLABILITY *of chordal* $(2,1)$*-graphs is* $\mathcal{NP}$*-complete.*

Sketch of the proof. Consider the graph H obtained by Construction 5 on a non-bipartite instance graph G of 3-COLORABILITY. We have $V(H) = V(G) \cup E(G) \cup V_2 \cup \{u\}$. Clearly, H is chordal and $(2,1)$ with $E(G)$ as a clique, $V(G) \cup \{u\}$ as a stable set, and V_2 as another stable set.

The key idea of the proof is that the pendant vertices V_2 force the maximal cliques incident to the vertices in $V(G) \cup \{u\}$ to be non-exposed. Notice that the tessellation used to cover the clique $E(G) \cup \{u\}$ cannot cover any other maximal clique with size greater than one incident to $V(G)$ vertices and $E(G)$ vertices in H. Therefore, there are only three remaining tessellations to cover those maximal cliques. This implies that G is 3-colorable if and only if H is 4-tessellable because if $uv \in E(G)$, then they receive different colors in a 3-coloring of G. The maximal clique containing vertices $E(G) \cup \{u\}$ and the maximal clique containing vertices $E(G) \cup \{v\}$ share a same neighborhood uv and must be covered by different tessellations. □

Construction 6. Let H' be a graph obtained from the graph of Construction 5 by transforming the stable set of $V(G)$ into a clique, removing one pendant vertex of each vertex of $V(G)$, and adding a vertex u' adjacent to all vertices of $V(G)$ with three new pendant vertices incident to it.

Clearly, H' from Construction 6 is a $(1,2)$-graph. Observe that H' is 4-tessellable if and only if H (from Theorem 6) is 4-tessellable. Therefore, 4-TESSELLABILITY is $\mathcal{NP}$-complete for $(1,2)$-graphs.

Next, we show a polynomial transformation from the $\mathcal{NP}$-complete problem NAE 3-SAT [7] to 3-TESSELLABILITY of diamond-free graphs with diameter five. This proof is given in two phases: given an instance I of NAE 3-SAT we construct a clique graph $K(G)$ for which we show that there is a 3-coloring of $K(G)$ if and only if I is satisfiable; subsequently, we show that there is a construction of a graph with diameter five G for which G is 3-tessellable if and only if $K(G)$ is 3-colorable.

Construction 7. Let $K(G)$ be a graph obtained from an instance of NAE 3-SAT as follows. For each variable v of I, include a P_2 with vertices v and $\overline{v}$ in $K(G)$. Moreover, add a vertex u adjacent to all P_2's vertices. And, for each clause $\{a \vee b \vee c\}$ of I, add a triangle with vertices T_a, T_b, T_c in $K(G)$ and three edges aT_a, bT_b, and cT_c.

Lemma 5. *Let* $K(G)$ *be obtained from Construction 7 on a* NAE 3-SAT *instance* I*. Then* $K(G)$ *is* 3*-colorable if and only if* I *is satisfiable.*

Proof. Note that, w.l.o.g., the color 1 given to the vertex u in a 3-coloring cannot be used in any vertex of a P_2. Moreover, each of the literal vertices v and $\overline{v}$ of a P_2 receives either the color 2 or 3. Assume w.l.o.g that a literal is true if its color is 2, and false otherwise.

If $K(G)$ is 3-colorable, then there are no three vertices connected to a clause's triangle with the same color. Otherwise, this color would not be used in the triangle vertices and $K(G)$ would not be 3-colorable, a contradiction. Therefore, the above assignment of values to literals would give a satisfiable solution to the instance.

Conversely, if I is satisfiable, then one may assign color 2 to each literal vertex which is true and color 3 to its negation. Moreover, vertex u receives color 1. Since there are no three literal vertices with the same color adjacent to the clause triangles, one may assign colors to the vertices of the triangles in a 3-coloring where a vertex of the triangle adjacent to a vertex with color 2 receives color 3, and a vertex adjacent to a vertex with color 3 receives color 2. The other vertex receives color 1. □

Next, we construct a graph G for which its clique graph is the $K(G)$ obtained from Construction 7.

Construction 8. Let G be obtained from its clique graph $K(G)$ (of Construction 7) as follows. For each clause's triangle in $K(G)$, add a star with three leaves in G, where each of those leaves represents a literal of this clause. Next, all P_2's triangles in $K(G)$ are represented in G by a clique C of size the number of P_2's. Each vertex of this clique C represents a variable of $K(G)$. For each vertex v of C include the edges of two other cliques (one for each literal of the variable v) composed by the leaves of the stars which represent the literals v and $\overline{v}$ and the vertex v of C, as depicted in Fig. 4.

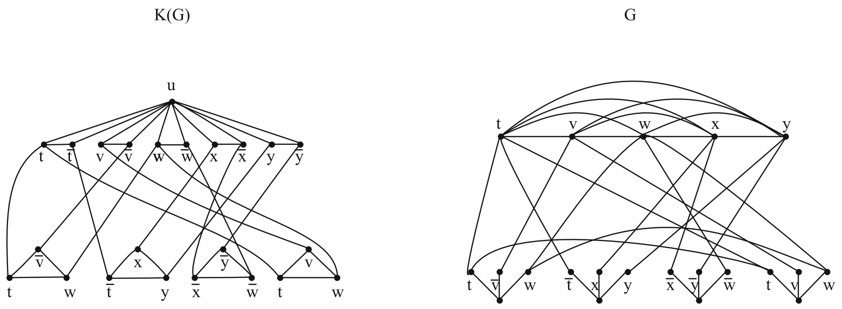


Fig. 4. Example of Construction 8.

Lemma 6. *Let $K(G)$ be obtained by Construction 7 on a* NAE 3-SAT *instance I and G be obtained by Construction 8 on $K(G)$. Then G is 3-tessellable if and only if $K(G)$ is 3-colorable.*

Proof. If G is 3-tessellable, we need one tessellation to cover the maximum clique whose size is the number of variables. Therefore, the other two tessellations are used by the other two maximal cliques (which represent the literals of each variable). Moreover, the star of three leaves of each clause also needs to be covered by 3 tessellations. Note that these maximal cliques represent vertices in $K(G)$ and the tessellations represent their colors. Therefore, $K(G)$ is 3-colorable.

If $K(G)$ is 3-colorable, then use these three colors as a guide to obtain a 3-tessellation of G, where each color class of $K(G)$ represents that these maximal cliques of G are covered by the tessellation which represents this color. □

Clearly, the graph G obtained from Construction 8 is diamond-free with diameter five. Therefore, by Lemmas 5 and 6, the next theorem follows.

Theorem 7. 3-TESSELLABILITY *of diamond-free graphs with diameter five is $\mathcal{NP}$-complete.*

4.2 2-Tessellability

Portugal [3] showed that a graph G is 2-tessellable if and only if $K(G)$ is a bipartite graph. Moreover, Peterson [10] showed that $K(G)$ is bipartite if and only if G is the line graph of a bipartite multigraph. Hence, determine if G is 2-tessellable is equivalent to verifying if G is the line graph of a bipartite multigraph.

Protti and Szwarcfiter [14] showed an $O(n^2m)$ time algorithm to decide if the clique graph of a given graph is bipartite. Moreover, Peterson [10] showed an $O(n^3)$ time algorithm to decide if G is the line graph of a bipartite multigraph.

The key idea of Peterson's algorithm is to group true twin vertices of a same clique of G, which represent multiedges in the bipartite multigraph H, where $G = L(H)$. Then, it removes all those true twin vertices in each group but one, and the resulting graph is a line graph of a bipartite simple graph if and only if $K(G)$ is a bipartite graph. To verify if a graph is a line graph of a bipartite graph, the Roussopoulos' linear-time algorithm is used [15].

We improve Peterson's algorithm [10], by showing a faster way to remove true twin vertices belonging to a clique of a graph using its modular decomposition. Throughout this section, we use notations of modules of a graph given in [16].

Let $\mathcal{F}$ be the family of bipartite multigraphs obtained by adding multiple edges to C_4, S_n or P_4. In order to make a modular decomposition of a graph G, we only consider graphs G which are not a line graph of a graph in $\mathcal{F}$. If G is a line graph of a graph in $\mathcal{F}$, we can consider this case separately, and easily achieve linear time. Note that there are bipartite multigraphs with a same line graph. Therefore, we only consider the ones which maximize the number of multiple edges. Moreover, we only consider connected graphs, since the tessellation cover number of a disconnected graph is the maximum among the parameter on its connected components.

Lemma 7. *Let H be a bipartite multigraph not in $\mathcal{F}$ and $L(H)$ be its line graph. Two edges e_1 and e_2 with same extremes in H represent vertices in a same maximal strong module of $L(H)$ with size less than $|V(L(H))|$.*

Lemma 8. *Let H be a bipartite multigraph not in $\mathcal{F}$ and $L(H)$ be its line graph. Any maximal strong module in a modular decomposition of $L(H)$ with size less than $|V(L(H))|$ induces a clique in $L(H)$.*

Theorem 8. 2-TESSELLABILITY *can be solved in linear time.*

Proof. First, we use McConnell and Spinrad's linear-time algorithm to obtain a modular decomposition of G. By Lemmas 7 and 8, we know that the strong modules in any modular decomposition of a line graph of a bipartite multigraph $H \notin \mathcal{F}$ induce cliques. Moreover, the vertices of these cliques in $L(H)$ are related to edges of H with same extremes.

Then, we check if each of at most $O(|V(G)|)$ strong modules induces cliques in G, which can be done in $O(|V(G)| + |E(G)|)$. Otherwise, we know that G is not a line graph of a bipartite multigraph. Next, we remove all true twins vertices in each strong modules but one, obtaining the graph G'. This step is related to remove all multiedges of H which share same extremes. Therefore, the graph G is a line graph of a bipartite multigraph H if the resulting graph G' is a line graph of a bipartite simple graph H'.

Finally, we use Roussopoulos' linear-time algorithm to determine if G' is a line graph, and if so, obtain its root graph H' for which G' is the line graph. Note that verifying to if H' is a bipartite graph can be done in linear time by using a breadth-first search (because the size of the root graph of G' is asymptotically bounded by the size of G'). □

5 Concluding Remarks and Discussion

We investigate extremal results on graph tessellation covers, which are fundamental for the development of quantum walks in the staggered model. These results help to understand the complexity of the unitary operators necessary to express the evolution of staggered quantum walks. We establish tight upper bounds for the tessellation cover number of a graph G related to $\chi'(G)$ and $\chi(K(G))$ and we determine graph classes which reach these extremal bounds. This study provides tools to distinguish several classes for which the t-TESSELLABILITY problem is efficiently tractable (bipartite, diamond-free K-perfect and {triangle, proper major}-free graphs) from others the problem is $\mathcal{NP}$-complete (planar, triangle-free, chordal $(2,1)$-graphs, $(1,2)$-graphs, and diamond-free graphs with diameter five). We also establish the t-TESSELLABILITY $\mathcal{NP}$-completeness for any fixed $t \geq 3$. Moreover, we improve to linear time the known algorithms to recognize 2-tessellable graphs [3], line graphs of bipartite multigraphs [14], and graphs G such that $K(G)$ is bipartite [10]. As a consequence, we establish an interesting complexity dichotomy between EDGE-COLORABILITY and t-TESSELLABILITY: EDGE-COLORABILITY of planar graphs with $\Delta \geq 8$ is in $\mathcal{P}$ [17], while t-TESSELLABILITY is $\mathcal{NP}$-complete (Theorem 4 replacing each of the four non external triangles that shares two vertices of external triangles by K_4's) and; EDGE-COLORABILITY of line graph of bipartite graphs is $\mathcal{NP}$-complete [18], while t-TESSELLABILITY is in $\mathcal{P}$ (Theorem 8). We have not managed yet to establish the same dichotomy to k-COLORABILITY OF CLIQUE GRAPH and t-TESSELLABILITY. We are currently trying to establish the hardness of the problem for $(1, 1)$-graphs (split graphs). However, the computational complexity for $(0, 2)$-graphs (complement of bipartite graph) is still widely open.

A question that naturally arises is whether every graph has a minimum tessellation cover such that every tessellation contains a maximal clique. Although we believe in most cases the answer is true, we have computationally found a surprising example of a graph, which is depicted in Fig. 5, with all minimum tessellation covers requiring a tessellation without maximal cliques. We are currently trying to establish an infinite family of graphs for which this property does not hold and to establish other graph classes where it holds. The computational verification was performed through a reduction from t-TESSELLABILITY problem to SET-COVERING problem (the description of SET-COVERING is available at [7]), where the finite set is the edge set of the input graph, and the family of subsets consists of the edge subsets corresponding to all possible tessellations of the input graph. Another interesting issue is that two minimum tessellation covers may present different quantum walk dynamics. Therefore, we intend to study the different tessellation covers using the same number of tessellations, which may result in simpler quantum walks and more efficient quantum algorithms. More recently, a general partition-based framework for quantum walks has been proposed [19].

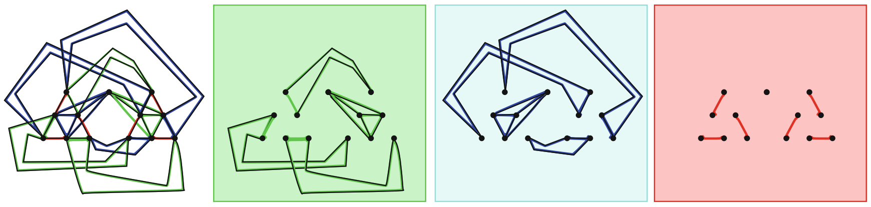


Fig. 5. 3-tessellable graph. Rightmost tessellation does not contain a maximal clique.

References

1. Venegas-Andraca, S.: Quantum walks: a comprehensive review. Quantum Inf. Process. **11**(5), 1015–1106 (2012)
2. Portugal, R., Santos, R.A.M., Fernandes, T.D., Gonçalves, D.N.: The staggered quantum walk model. Quantum Inf. Process. **15**(1), 85–101 (2016)
3. Portugal, R.: Staggered quantum walks on graphs. Phys. Rev. A **93**, 062335 (2016)
4. Abreu, A., Cunha, L., Fernandes, T., de Figueiredo, C., Kowada, L., Marquezino, F., Posner, D., Portugal, R.: Bounds and complexity for the tessellation problem. Mat. Contemp. (2017, accepted)
5. Szwarcfiter, J.L.: A survey on clique graphs. In: Reed, B.A., Sales, C.L. (eds.) Recent Advances in Algorithms and Combinatorics. CBMOS, pp. 109–136. Springer, New York (2003)
6. West, D.: Introduction to Graph Theory. Pearson, London (2000)

7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman Co., San Francisco (1979)
8. Zatesko, L.M., Carmo, R., Guedes, A.L.P.: Edge-colouring of triangle-free graphs with no proper majors. In: II Encontro de Teoria da Computação, pp. 71–74 (2017)
9. Koreas, D.P.: The NP-completeness of chromatic index in triangle free graphs with maximum vertex of degree 3. Appl. Math. Comput. **83**(1), 13–17 (1997)
10. Peterson, D.: Gridline graphs: a review in two dimensions and an extension to higher dimensions. Discrete Appl. Math. **126**(2–3), 223–239 (2003)
11. Bonomo, F., Durán, G., Groshaus, M., Szwarcfiter, J.: On clique-perfect and K-perfect graphs. Ars Comb. **80**, 97–112 (2006)
12. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1988)
13. Bodlaender, H., Kloks, T., Richard, B., van Leeuwen, J.: Approximation for lambda-colorings of graphs. Comput. J. **47**, 1–12 (2004)
14. Protti, F., Szwarcfiter, J.L.: Clique-inverse graphs of bipartite graphs. J. Comb. Math. Comb. Comput. **40**, 193–203 (2002)
15. Roussopoulos, N.D.: A max {m, n} algorithm for determining the graph H from its line graph G. Inf. Process. Lett. **2**, 108–112 (1973)
16. McConnell, R., Spinrad, J.: Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, vol. 2, pp. 536–545 (1994)
17. Sanders, D.P., Zhao, Y.: Planar graphs of maximum degree seven are class I. J. Comb. Theory B **83**(2), 201–212 (2001)
18. Cai, L., Ellis, J.A.: NP-completeness of edge-colouring some restricted graphs. Discrete Appl. Math. **30**(1), 15–27 (1991)
19. Konno, N., Portugal, R., Sato, I., Segawa, E.: Partition-based discrete-time quantum walks

Approximate Correlation Clustering Using Same-Cluster Queries

Nir Ailon[1], Anup Bhattacharya[2](✉), and Ragesh Jaiswal[2]

[1] Technion, Haifa, Israel
nailon@cs.technion.ac.il
[2] Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India
{anupb,rjaiswal}@cse.iitd.ac.in

Abstract. Ashtiani et al. (NIPS 2016) introduced a semi-supervised framework for clustering (SSAC) where a learner is allowed to make *same-cluster* queries. More specifically, in their model, there is a query oracle that answers queries of the form "*given any two vertices, do they belong to the same optimal cluster?*". In many clustering contexts, this kind of oracle queries are feasible. Ashtiani et al. showed the usefulness of such a query framework by giving a polynomial time algorithm for the k-means clustering problem where the input dataset satisfies some *separation* condition. Ailon et al. extended the above work to the approximation setting by giving an efficient $(1+\varepsilon)$-approximation algorithm for k-means for any small $\varepsilon > 0$ and any dataset within the SSAC framework. In this work, we extend this line of study to the *correlation clustering* problem. Correlation clustering is a graph clustering problem where pairwise similarity (or dissimilarity) information is given for every pair of vertices and the objective is to partition the vertices into clusters that minimise the disagreement (or maximises agreement) with the pairwise information given as input. These problems are popularly known as MinDisAgree and MaxAgree problems, and MinDisAgree$[k]$ and MaxAgree$[k]$ are versions of these problems where the number of optimal clusters is at most k. There exist *Polynomial Time Approximation Schemes* (PTAS) for MinDisAgree$[k]$ and MaxAgree$[k]$ where the approximation guarantee is $(1+\varepsilon)$ for any small ε and the running time is polynomial in the input parameters but exponential in k and $1/\varepsilon$. We get a significant running time improvement within the SSAC framework at the cost of making a small number of same-cluster queries. We obtain an $(1+\varepsilon)$-approximation algorithm for any small ε with running time that is polynomial in the input parameters and also in k and $1/\varepsilon$. We also give non-trivial upper and lower bounds on the number of same-cluster queries, the lower bound being based on the *Exponential Time Hypothesis* (ETH). Note that the existence of an efficient algorithm for MinDisAgree$[k]$ in the SSAC setting exhibits the power of same-cluster

Nir Ailon acknowledges the generous support of ISF grant number 2021408.
Anup Bhattacharya acknowledges the support of TCS fellowship at IIT Delhi.
Ragesh Jaiswal acknowledges the support of ISF-UGC India-Israel Grant 2014.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 14–27, 2018.
https://doi.org/10.1007/978-3-319-77404-6_2

queries since such polynomial time algorithm (polynomial even in k and $1/\varepsilon$) is not possible in the classical (non-query) setting due to our conditional lower bounds. Our conditional lower bound is particularly interesting as it not only establishes a lower bound on the number of same cluster queries in the SSAC framework but also establishes a conditional lower bound on the running time of any $(1+\varepsilon)$-approximation algorithm for MinDisAgree$[k]$.

1 Introduction

Correlation clustering is a graph clustering problem where we are given similarity or dissimilarity information for pairs of vertices. The input is a graph G on n vertices. Edges of G are labeled as similar (positive) or dissimilar (negative). The clustering objective is to partition the vertices into clusters such that edges labeled 'positive' remain within clusters and 'negative' edges go across clusters. However, this similarity/dissimilarity information may be inconsistent with this objective. For example, there may exist vertices u, v, w such that edges (u, v) and (u, w) are labeled 'positive' whereas edge (v, w) is labeled 'negative'. In this case, it is not possible to come up with a clustering of these 3 vertices that would *agree* with all the edge labels. The objective of correlation clustering is to come up with a clustering that minimises disagreement or maximises agreement with the edge labels given as input. The minimisation version of the problem, known as MinDisAgree, minimises the sum of the number of negative edges present inside clusters and the number of positive edges going across clusters. Similarly, the maximisation version is known as MaxAgree where the objective is to maximise the sum of the number of positive edges present inside clusters and the number of negative edges going across clusters. Unlike k-means or k-median clustering, in correlation clustering, there is no restriction on the number of clusters formed by the optimal clustering. When the number of optimal clusters is given to be at most k, these problems are known as MinDisAgree$[k]$ and MaxAgree$[k]$ respectively.

Bansal et al. [8] gave a constant factor approximation algorithm for MinDisAgree and a PTAS for MaxAgree. Subsequently, Charikar et al. [9] improved the approximation guarantee for MinDisAgree to 4, and showed that MinDisAgree is APX-hard. These results are for correlation clustering on complete graphs as it is known for general graphs, it is at least as hard as *minimum multicut* problem [9]. Since MinDisAgree is APX-hard [9], additional assumptions were introduced for better results. For example [15,17] studied MinDisAgree where the input is noisy and comes from a semi-random model. When k is given as part of the input, Giotis and Guruswami [12] gave a PTAS for MinDisAgree$[k]$.

Recently there have been some works [2,7] with a *beyond-worst case* flavour where polynomial time algorithms for NP-hard problems have been designed under some stability assumptions. Ashtiani et al. [3] considered one such stability assumption called γ-*margin*. They introduced a semi-supervised active learning (SSAC) framework and within this framework, gave a probabilistic polynomial time algorithm for k-means on datasets that satisfy the γ-margin property. More specifically, their SSAC framework involves a query oracle that answers queries of the form *"given any two vertices, do they belong to the same optimal cluster?"*. The query oracle responds with a Yes/No answer where these answers are assumed to be consistent with some fixed optimal solution. In this framework, they studied the query complexity for polynomial time algorithms for k-means on datasets satisfying the γ-margin property. Ailon et al. [1] extended this work to study query complexity bounds for $(1+\varepsilon)$-approximation for k-means in SSAC framework for any small $\varepsilon > 0$ without any stability assumption on the dataset. They gave almost matching upper and lower bounds on the number of queries for $(1 + \varepsilon)$-approximation of k-means problem in SSAC framework.

In this work, we study $\mathsf{MinDisAgree}[k]$ in the SSAC framework, where the optimal clustering has at most k clusters and give upper and lower bounds on the number of same-cluster queries for $(1+\varepsilon)$-approximation for correlation clustering for any $\varepsilon > 0$. We also give upper bounds for $\mathsf{MaxAgree}[k]$. Our algorithm is based on the PTAS by Giotis and Guruswami [12] for $\mathsf{MinDisAgree}[k]$. The algorithm by Giotis and Guruswami involves random sampling a subset S of vertices and considers all possible ways of partitioning S into k clusters $S = \{S_1, \ldots, S_k\}$, and for every such k-partitioning, clusters the rest of the vertices greedily. Every vertex $v \in V \setminus S$ is assigned a cluster S_j that maximizes its agreement with the edge labels. Their main result was the following.

Theorem 1 (Giotis and Guruswami [12]). *For every $k \geq 2$, there is a PTAS for $\mathsf{MinDisAgree}[k]$ with running time $n^{O(9^k/\varepsilon^2)} \log n$.*

Since Giotis and Guruswami considered all possible ways of partitioning subset S into k clusters, their running time has exponential dependence on k. Here, we make the simple observation that within the SSAC framework we can overcome this exponential dependence on k by making same-cluster queries to the oracle. The basic idea is to randomly sample a subset S of vertices as before and partition it optimally into k clusters by making same-cluster queries to the oracle. Note that by making at most $k|S|$ same-cluster queries, one can partition S optimally into k clusters. Once we have subset S partitioned as in the optimal clustering (a key step needed in the analysis of Giotis and Guruswami) we follow their algorithm and analysis for $(1 + \varepsilon)$-approximation for $\mathsf{MinDisAgree}[k]$. Here is our main result for $\mathsf{MinDisAgree}[k]$ in the SSAC framework. We obtain similar results for $\mathsf{MaxAgree}[k]$.

Theorem 2 (Main result: Upper bound). *Let $\varepsilon > 0$ and $k \geq 2$. There is a* (randomized) *algorithm in the SSAC framework for* $\mathsf{MinDisAgree}[k]$ *that uses* $O\left(\frac{k^{14} \log k \log n}{\varepsilon^6}\right)$ *same-cluster queries, runs in time* $O(\frac{nk^{14} \log k \log n}{\varepsilon^6})$ *and outputs a* $(1 + \varepsilon)$*-approximate solution with high probability.*

We complement our upper bound result by providing a lower bound on the number of queries in the SSAC framework for any efficient $(1+\varepsilon)$-approximation algorithm for $\mathsf{MinDisAgree}$ for any $\varepsilon > 0$. Our lower bound result is conditioned on the *Exponential Time Hypothesis* (ETH hypothesis) [13,14]. Our lower bound result implies that the number of queries is depended on the number of optimal clusters k. Our main result with respect to query lower bound is given as follows.

Theorem 3 (Main result: Lower bound). *Given that Exponential Time Hypothesis* (ETH) *holds, there exists a constant* $\delta > 0$ *such that any* $(1+\delta)$*-approximation algorithm for* MinDisAgree$[k]$ *in the SSAC framework that runs in polynomial time makes* $\Omega(\frac{k}{poly \log k})$ *same-cluster queries.*

Exponential Time Hypothesis is the following statement regarding the hardness of the 3-SAT problem.

> *Exponential Time Hypothesis (ETH)* [13,14]: There does not exist an algorithm that can decide whether any 3-SAT formula with m clauses is satisfiable with running time $2^{o(m)}$.

Note that our query lower bound result is a simple corollary of the following theorem that we prove.

Theorem 4. *If the Exponential Time Hypothesis (ETH) holds, then there exists a constant* $\delta > 0$ *such that any* $(1+\delta)$*-approximation algorithm for* MinDisAgree$[k]$ *requires* $2^{\Omega(\frac{k}{poly \log k})}$ *time.*

The above lower bound statement may be of independent interest. It was already known that MinDisAgree is APX-hard. Our result is a non-trivial addition to the understanding of the hardness of the correlation clustering problem. Given that our query upper bound result is through making simple observations in the algorithms of Giotis and Guruswami, our lower bound results may be regarded as the primary contribution of this work. So, we first give our lower bound results in the next section and the upper bound results in Sect. 3. However, before we start discussing our results, here is a brief discussion on the related works.

Related Works. There have been numerous works on clustering problems in semi-supervised settings. Balcan and Blum [5] proposed an interactive framework for clustering which use 'split/merge' queries. In this framework, given any arbitrary clustering $C = \{C_1, C_2, \ldots, \}$ as query, oracle specifies some cluster C_l should be split or clusters C_i and C_j should be merged. Awasthi et al. [4] developed a local clustering algorithm which uses these split/merge queries. *One versus all* queries for clustering were studied by Voevodski et al. [19]. The oracle, on a query $s \in X$, returns distances from s to all points in X. The authors provided a clustering, close to optimal k-median clustering, with only $O(k)$ such queries on instances satisfying (c, ε)-approximation stability property [6]. Fomin *et al.* [11] gave a conditional lower bound for the *cluster editing* problem which can also be stated as a decision version of the correlation clustering problem. In the p-cluster editing problem, given a graph G and a budget B, and an integer p, the objective is to decide whether G can be transformed into a union of p clusters (disjoint cliques) using at most B edge additions and deletions. Assuming ETH, they showed that there exists $p = \Theta(k^{\omega})$ for some $0 \leq \omega \leq 1$ such that there is no algorithm that decides in time $2^{o(\sqrt{pB})} \cdot n^{O(1)}$ whether G can be transformed into a union of p cliques using at most B adjustments (edge additions and deletions). It is not clear whether their exact reduction can be modified into an approximation preserving reduction to obtain results similar to what we have here. Mazumdar and Saha [18] studied correlation clustering problem in a similar setting where edge similarity and dissimilarity information are assumed to be coming from two distributions. Given such an input, they studied the *cluster recovery* problem in SSAC framework, and gave upper and lower bounds on the query complexity. Their lower bound results are information theoretic in nature. We are, however, interested in the approximate solutions for the correlation clustering problem.

2 Query Lower Bounds

In this section, we obtain a lower bound on the number of same-cluster queries that any FPTAS within the SSAC framework needs to make for the problem MinDisAgree$[k]$. We derive a conditional lower bound for the minimum number of queries under the Exponential Time Hypothesis (ETH) assumption. Some such conditional lower bound results based on ETH can be found in [16]. We prove the following main theorem in this section.

Theorem 5. *If the Exponential Time Hypothesis (ETH) holds, then there exists a constant $\delta > 0$ such that any $(1+\delta)$-approximation algorithm for* MinDisAgree$[k]$ *requires $2^{\Omega(\frac{k}{poly \log k})}$ time.*

The above theorem gives a Proof of Theorem 3.

Proof (Proof of Theorem 3). Let us assume that there exists a query-FPTAS that makes only $o(\frac{k}{\text{poly} \log k})$ same-cluster queries. Then, by considering all possible answers for these queries and picking the best solution, one can solve the problem in $2^{o(\frac{k}{\text{poly} \log k})}$ time which contradicts Theorem 5.

In the remaining section, we give the Proof of Theorem 5. First, we state the ETH hypothesis. Our lower bound results are derived assuming this hypothesis.

Hypothesis 1 *(Exponential Time Hypothesis (ETH) [13,14])*: There does not exist an algorithm that decides whether any 3-SAT formula with m clauses is satisfiable with running time $2^{o(m)}$.

Since we would like to obtain lower bounds in the approximation domain, we will need a *gap* version of the above ETH hypothesis. The following version of the PCP theorem would be very useful in obtaining a gap version of ETH.

Theorem 6 (Dinur's PCP Theorem [10]). *For some constants $\varepsilon, d > 0$, there exists a polynomial-time reduction that takes a* 3-SAT *formula ψ with m clauses as input and produces one* E3-*SAT*[1] *formula ϕ with $m' = O(m poly \log m)$ clauses such that*

[1] Every clause in an E3-SAT formula has exactly 3 literals.

– *if ψ is satisfiable, then ϕ is satisfiable, and*
– *if ψ is unsatisfiable, then $val(\phi) \leq 1 - \varepsilon$, and*
– *each variable in ϕ appears in at most d clauses.*

where $val(\phi)$ is the maximum fraction of clauses of ϕ which are satisfiable by any assignment.

The hypothesis below follows from ETH and the above Theorem 6, and will be useful for our analysis.

Hypothesis 2: There exists constants $\varepsilon, d > 0$ such that the following holds: There does not exist an algorithm that, given a E3-SAT formula ψ with m clauses and each variable appearing in at most d clauses, distinguishes whether ψ is satisfiable or $val(\psi) \leq (1-\varepsilon)$, and runs in time better than $2^{\Omega\left(\frac{m}{\text{poly}\log m}\right)}$.

The lemma given below trivially follows from Dinur's PCP Theorem 6.

Lemma 1. *If Hypothesis 1 holds, then so does Hypothesis 2.*

We now give a reduction from the gap version of the E3-SAT problem to the gap version of the NAE3-SAT problem. A problem instance of NAE3-SAT consists of a set of clauses (each containing exactly 3 literals) and a clause is said to be satisfied by an assignment iff at least one and at most two literals in the clause is true (NAE stands for "Not All Equal"). For any instance ϕ, we define $val'(\phi)$ to be the maximum fraction of clauses that can be satisfied in the "not all equal" sense by an assignment. Note that this is different from $val(\phi)$ which is equal to the maximum fraction of clauses that can be satisfied (in the usual sense). First, we reduce E3-SAT to NAE6-SAT and then NAE6-SAT to NAE3-SAT.

Lemma 2. *Let $0 < \varepsilon < 1$ and $d > 1$. There is a polynomial time reduction that given an instance ψ of* E3-SAT *with m clauses with each variable appearing in at most d clauses, produces an instance ϕ of* NAE6-SAT *with $4m$ clauses such that*

1. *If $val(\psi) = 1$, then $val'(\phi) = 1$, and*
2. *If $val(\psi) \leq (1 - \varepsilon)$, then $val'(\phi) \leq (1 - \varepsilon/4)$, and*
3. *Each variable in ϕ appears in at most $4d$ clauses.*

Proof. We construct ϕ in the following manner: for every variable x_i in ψ, we introduce two variables y_i and z_i. We will use $x_i = 1$ iff $y_i \neq z_i$ for every i in our reduction. For every clause (l_i, l_j, l_k) (with l_i, l_j, l_k being literals), we introduce the following four NAE clauses in ϕ:

$$(p_i, q_i, p_j, q_j, p_k, q_k), (p_i, q_i, p_j, q_j, \bar{p}_k, \bar{q}_k), (p_i, q_i, \bar{p}_j, \bar{q}_j, p_k, q_k), (p_i, q_i, \bar{p}_j, \bar{q}_j, \bar{p}_k, \bar{q}_k)$$

For any index (say i), if $l_i = x_i$ (that is, the variable is in the positive form), then $p_i = y_i$ and $q_i = z_i$. On the other hand, if $l_i = \bar{x}_i$, then $p_i = y_i$ and $q_i = \bar{z}_i$. So for example, for the clause $(x_2, \bar{x}_7, x_9)$ in ψ, we have the following four clauses:

$$(y_2, z_2, y_7, \bar{z}_7, y_9, z_9), (y_2, z_2, y_7, \bar{z}_7, \bar{y}_9, \bar{z}_9), (y_2, z_2, \bar{y}_7, z_7, y_9, z_9), (y_2, z_2, \bar{y}_7, z_7, \bar{y}_9, \bar{z}_9)$$

Note that property (3) of the lemma holds due to our construction. For property (1), we argue that for any satisfying assignment for ψ, the assignment of variables in ϕ as per the rule $x_i = 1$ iff $y_i \neq z_i$ is a satisfying assignment of ψ (in the NAE sense). This is because for every literal l that makes a clause in ψ true, the two corresponding copies p and q satisfies all the four clauses (in the NAE sense). For property (2), we prove the contrapositive. Suppose there is an assignment to the y, z variables in ϕ that satisfies at least $(1 - \varepsilon/4)$ fraction of the clauses. We will argue that the assignment to the variables of ψ as per the rule $x_i = 1$ iff $y_i \neq z_i$ satisfies at least $(1 - \varepsilon)$ fraction of clauses of ψ. First, note that for every set of 4 clauses in ϕ created from a single clause of ψ, either 3 of them are satisfied or all four are satisfied (whatever the assignment of the variable be). Let m_1 be the number of these 4-sets where all 4 clauses are satisfied and let m_2 be the number of these 4-sets where 3 clauses are satisfied, where $m = m_1 + m_2$. Then we have $4m_1 + 3m_2 \geq (1 - \varepsilon/4) \cdot (4m)$ which implies that $m_1 \geq (1 - \varepsilon)m$. Note that for any of the 4-sets where all 4 clauses are satisfied, the corresponding clause in ψ is satisfied with respect to the assignment as per rule $x_i = 1$ iff $y_i \neq z_i$ (since at least one the p, q pairs will have opposite values). So, the fraction of the clauses satisfied in ψ is at least $\frac{m_1}{m} \geq (1 - \varepsilon)$.

Lemma 3. *Let $0 < \varepsilon < 1$ and $d > 1$. There is a polynomial time reduction that given an instance ψ of* NAE6-SAT *with m clauses and with each variable appearing in at most d clauses, produces an instance ϕ of* NAE3-SAT *with $4m$ clauses such that:*

1. *If $val'(\psi) = 1$, then $val'(\phi) = 1$, and*
2. *If $val'(\psi) \leq (1 - \varepsilon)$, then $val'(\phi) \leq (1 - \varepsilon/4)$.*
3. *Each variable in ϕ appears in at most $\max(d, 2)$ clauses.*

Proof. For every clause $C_i = (a_i, b_i, c_i, d_i, e_i, f_i)$ in ψ, we construct the following four clauses in ϕ (let us call it a 4-set): $(a_i, b_i, x_i), (\bar{x}_i, c_i, y_i), (\bar{y}_i, d_i, z_i), (\bar{z}_i, e_i, f_i)$, introducing new variables x_i, y_i, z_i. Property (3) trivially holds for this construction. For every satisfying assignment for ψ, there is a way to set the clause variables x_i, y_i, z_i for every i such that all four clauses in the 4-set corresponding to clause C_i are satisfied. So, property (1) holds. We show property (2) using contraposition. Consider any assignment of ϕ that satisfies at least $(1-\varepsilon/4)$ fraction of the clauses. Let m_j denote the number of 4-sets such that as per this assignment j out of 4 clauses are satisfied. Then, we have $\sum_{j=0}^{4} j \cdot m_j \geq (1 - \varepsilon/4) \cdot (4m)$. This implies that: $3 \cdot \sum_{j=0}^{3} m_j + 4m_4 \geq (1 - \varepsilon/4) \cdot (4m)$ which implies that $m_4 \geq (1 - \varepsilon)m$. Now, note that for any 4-set such that all four clauses are satisfied, the corresponding clause in ψ is satisfied by the same assignment to the variables. This implies that there is an assignment that makes at least $(1 - \varepsilon)$ fraction of clauses true in ψ.

We come up with the following hypothesis which holds given that Hypothesis 2 holds, and is crucial for our analysis.

Hypothesis 3: There exists constants $\varepsilon, d > 0$ such that the following holds: There does not exist an algorithm that, given a NAE3-SAT formula

ψ with m clauses with each variable appearing in at most d clauses, distinguishes whether $val'(\psi) = 1$ or $val'(\psi) \leq (1 - \varepsilon)$, and runs in time better than $2^{\Omega\left(\frac{m}{\text{poly}\log m}\right)}$.

The lemma given below follows easily from the Lemmas 2 and 3 above.

Lemma 4. *If Hypothesis 2 holds, then so does Hypothesis 3.*

We now give a reduction from the gap version of NAE3-SAT to the gap version of *monotone* NAE3-SAT that has no negative variables. Note that because of the NAE (not all equal) property, setting all variables to 1 does not necessarily satisfy the formula.

Lemma 5. *Let $0 < \varepsilon < 1$ and $d > 1$. There is a polynomial time reduction that given an instance ψ of* NAE3-SAT *with m clauses and with each variable appearing in at most d clauses, produces an instance ϕ of* monotone NAE3-SAT *with $O(m)$ clauses such that:*

1. *If $val'(\psi) = 1$, then $val'(\phi) = 1$, and*
2. *If $val'(\psi) \leq (1 - \varepsilon)$, then $val'(\phi) \leq (1 - \frac{\varepsilon}{1+12d})$.*
3. *Each variable in ϕ appears in at most $4d$ clauses.*

Proof. We construct ϕ in the following manner: Substitute all positive literals of the variable x_i with y_i and all negative literals with z_i for new variables y_i, z_i. Also, for every variable x_i, add the following $4d$ clauses:

$$\{(y_i, z_i, t_i^j), (y_i, z_i, u_i^j), (y_i, z_i, v_i^j), (t_i^j, u_i^j, v_i^j)\}_{j=1}^{d}$$

where t_i^j, u_i^j, v_i^j for $1 \leq j \leq d$ are new variables. Note that the only way to satisfy all the above clauses is to have $y_i \neq z_i$. Let m' denote the total number of clauses in ϕ. So, $m' = m + 4dn$. Also, from the construction, each variable in ϕ appears in at most $4d$ clauses. This proves property (3). Property (1) follows from the fact that for any satisfying assignment for ψ, there is a way to extend this assignment to variables in ϕ such that all clauses are satisfied. For all i, $y_i = x_i$ and $z_i = \bar{x}_i$. All the new variables t, u, v can be set so as to make all the new clauses satisfied.

We argue property (2) using contraposition. Suppose there is an assignment to variables in ϕ that makes at least $(1 - \varepsilon/(1 + 12d))$ fraction of clauses satisfied. First, note that there is also an assignment that makes at least $(1 - \varepsilon/(1 + 12d))$ fraction of the clauses satisfied and in which for all i, $y_i \neq z_i$. This is because $3d$ out of $4d$ of the following clauses can be satisfied when $y_i = z_i$:

$$\{(y_i, z_i, t_i^j), (y_i, z_i, u_i^j), (y_i, z_i, v_i^j), (t_i^j, u_i^j, v_i^j)\}_{j=1}^{d}$$

However, if we flip one of y_i, z_i, then the number of above clauses satisfied can be $4d$ and we might lose out on at most d clauses since a variable appears in at most d clauses in ψ. Let m' be the number of clauses corresponding to the original clauses that are satisfied with this assignment. So, we have $m' + 4nd > (1 - \frac{\varepsilon}{(1+12d)})(m + 4nd)$ which gives:

$$m' > (1-\varepsilon)m + \frac{12md\varepsilon}{1+12d} - \frac{4nd\varepsilon}{1+12d} \geq (1-\varepsilon)m \quad (\text{since } 3m \geq n)$$

This completes the proof of the lemma.

Next we propose a hypothesis which holds given that Hypothesis 3 holds.

Hypothesis 4: There exists constants $\varepsilon, d > 0$ such that the following holds: There does not exist an algorithm that, given a monotone NAE3-SAT formula ψ with m clauses with each variable appearing in at most d clauses, distinguishes whether $val'(\psi) = 1$ or $val'(\psi) \leq (1-\varepsilon)$, and runs in time better than $2^{\Omega\left(\frac{m}{\text{poly}\log m}\right)}$.

The lemma below follows easily from Lemma 5 mentioned in above.

Lemma 6. *If Hypothesis 3 holds, then so does Hypothesis 4.*

We provide a reduction from the gap version of monotone NAE3-SAT to a gap version of 2-colorability of 3-uniform bounded degree hypergraph.

Lemma 7. *Let $0 < \varepsilon < 1$ and $d > 1$. There exists a polynomial time reduction that given a monotone NAE3-SAT instance ψ with m clauses and with every variable appearing in at most d clauses, outputs an instance H of 3-uniform hypergraph with $O(m)$ vertices and hyperedges and with bounded degree d such that if ψ is satisfiable, then H is 2-colorable, and if at most $(1-\varepsilon)$-fraction of clauses of ψ are satisfiable, then any 2-coloring of H would have at most $(1-\varepsilon)$-fraction of edges that are bichromatic.*

Proof. The reduction constructs a hypergraph $H(V, E)$ as follows. The set of vertices V correspond to the set of variables (all of them positive literals) of the monotone NAE3-SAT instance ψ. The set of edges E correspond to the set of clauses all of which have 3 literals, and therefore every hyperedge is of size 3. The resulting hypergraph is 3-uniform, and since every variable appears in at most d clauses, the hypergraph H is of bounded degree d, and $|V| = O(m)$ and $|E| = O(m)$. If there exists a satisfying assignment for ψ, then every edge in H is bichromatic and the hypergraph would be 2-colorable, and if at most $(1-\varepsilon)$-fraction of clauses are satisfiable by any assignment, then at most $(1-\varepsilon)$-fraction of edges of H are bichromatic.

Next, we devise a hypothesis which holds given that Hypothesis 4 holds.

Hypothesis 5: There exists constants $\varepsilon, d > 0$ such that the following holds: There does not exist an algorithm that, given a 3-uniform hypergraph H with m vertices and where every vertex has degree at most d, distinguishes whether H is bichromatic or at most $(1-\varepsilon)$-fraction of edges are bichromatic, and runs in time better than $2^{\Omega\left(\frac{m}{\text{poly}\log m}\right)}$.

The lemma below follows easily from Lemma 7 above.

Lemma 8. *If Hypothesis 4 holds, then so does Hypothesis 5.*

We now give a reduction from 2-colorability in 3-uniform hypergraph H with constant bounded degree to a correlation clustering instance on a complete graph G. We use the reduction as given in [9] for our purposes.

Lemma 9 ([9]). *Let $\varepsilon, d > 0$. There is a polynomial-time reduction that given a 3-uniform hypergraph $H(V, E)$ with m vertices and where each vertex appears in at most d hyperedges, outputs an instance of the correlation clustering problem where the graph $G(V', E')$ has $N = O(m)$ vertices and $M = 2N$ edges with edges in E' are labeled as 'positive' and all the other edges in the complete graph on V' vertices are labeled as 'negative' such that the following holds:*

1. *If H is 2-colorable, then the cost of the optimal correlation clustering is $M - N$, and*
2. *If at most $(1 - \varepsilon)$-fraction of hyperedges of H are bi-chromatic, then the optimal cost of correlation clustering is at least $M - (1 - \delta)N$, where δ is some constant.*

Next, we propose a hypothesis which holds given that Hypothesis 5 holds.

Hypothesis 6: There exists constants $\varepsilon > 0$ such that the following holds: There does not exist a $(1 + \varepsilon)$-factor approximation algorithm for the MinDisAgree$[k]$ problem that runs in time better than $poly(n) \cdot 2^{\Omega(\frac{k}{poly \log k})}$.

The lemma below follows easily from Lemma 9 given above.

Lemma 10. *If Hypothesis 5 holds, then so does Hypothesis 6.*

Finally, the Proof of Theorem 5 follows from chaining together Lemmas 1, 4, 6, 8, and 10.

3 Algorithms for MaxAgree$[k]$ and MinDisAgree$[k]$ in SSAC Framework

In this section, we give $(1 + \varepsilon)$-approximation algorithms for the MaxAgree$[k]$ and MinDisAgree$[k]$ problems within the SSAC framework for any $\varepsilon > 0$.

3.1 MaxAgree$[k]$

In this section, we will discuss a query algorithm that gives $(1+\varepsilon)$-approximation to the MaxAgree$[k]$ problem. The algorithm that we will discuss is closely related to the non-query algorithm for MaxAgree$[k]$ by Giotis and Guruswami. See Algorithm **MaxAg**(k, ε) in [12]. In fact, except for a few changes, this section will look extremely similar to Sect. 3 in [12]. Given this, it will help if we mention the high-level idea of the Giotis-Guruswami algorithm and point out the changes that can be made within the SSAC framework to obtain the desired result. The algorithm of Giotis and Guruswami proceeds in m iterations, where $m = O(1/\varepsilon)$. The given dataset V is partitioned into m equal parts $V^1, \ldots, V^m$, and in the i^{th} iteration, points in V^i are assigned to one of the k clusters. In order to cluster V^i in the i^{th} iteration, the algorithm samples a set of data points S^i, and for all possible k-partitions of S^i, it checks the agreement of a point $v \in V^i$ with the k clusters of S^i. Suppose for a particular clustering $S_1^i, \ldots, S_k^i$ of S^i, the agreement of vertices in V^i is maximised. Then the vertices in V^i are clustered by placing them into the cluster that maximises their agreement with respect to $S_1^i, \ldots, S_k^i$. Trying out all possible k-partitions of S^i is an expensive operation in the Giotis-Guruswami algorithm (since the running time becomes $\Omega(k^{|S^i|})$). This is where the same-cluster queries help. Instead of trying out all possible k-partitions of S^i, we can make use of the same-cluster queries to find a *single* appropriate k-partition of S^i in the i^{th} iteration. This is the clustering that matches the "hybrid" clustering of Giotis and Guruswami. So, the running time of the i^{th} iteration improves from $O(k^{|S^i|})$ to $O(k \cdot |S^i|)$. Moreover, the number of same-cluster queries made in the i^{th} iteration is $k \cdot |S^i|$, thus making the total number of same-cluster queries to be $O(\frac{k}{\varepsilon} \cdot |S^i|)$. The theorem is given below. The details of the proof of this theorem is not given since it trivially follows from Giotis and Guruswami (see Theorem 3.2 in [12]).

Theorem 7. *There is a query algorithm* QueryMaxAg *that behaves as follows: On input ε, δ and a labelling $\mathcal{L}$ of the edges of a complete graph G with n vertices, with probability at least $(1-\delta)$, algorithm* QueryMaxAg *outputs a k clustering of the graph such that the number of agreements induced by this k-clustering is at least $OPT - \varepsilon n^2/2$, where OPT is the optimal number of agreements induced by any k-clustering of G. The running time of the algorithm is $O\left(\frac{nk}{\varepsilon^3} \log \frac{k}{\varepsilon^2 \delta}\right)$. Moreover, the number of same-cluster queries made by* QueryMaxAg *is $O\left(\frac{k}{\varepsilon^3} \log \frac{k}{\varepsilon^2 \delta}\right)$.*

Using the simple observation that $OPT \geq n^2/16$ (see Proof of Theorem 3.1 in [12]), we get that the above query algorithm gives $(1+\varepsilon)$-approximation guarantee in the SSAC framework.

3.2 MinDisAgree[k]

In this section, we provide a $(1+\varepsilon)$-approximation algorithm for the MinDisAgree[k] for any small $\varepsilon > 0$. Giotis and Guruswami [12] provided a $(1+\varepsilon)$-approximation algorithm for MinDisAgree[k]. In this work, we extend their algorithm to make it work in the SSAC framework with the aid of same-cluster queries, and thereby improve the running time of the algorithm considerably. Our query algorithm will be closely based on the non-query algorithm of Giotis and Guruswami. In fact, except for a small (but crucial) change, the algorithms are the same. So, we begin by discussing the main ideas and the result by Giotis and Guruswami.

Lemma 11 (Theorem 4.7 in [12]). *For every $k \geq 2$ and $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm for* MinDisAgree[k] *with running time $n^{O(9^k/\varepsilon^2)} \log n$.*

The algorithm by Giotis and Guruswami builds on the following ideas. First, from the discussion in the previous section, we know that there is a FPTAS within SSAC framework for $\mathsf{MaxAgree}[k]$. Therefore, unless OPT, the optimal value for $\mathsf{MinDisAgree}[k]$ is small (OPT$= \gamma n^2$, for some small $\gamma > 0$), the complement solution for $\mathsf{MaxAgree}[k]$ would give a valid $(1+\varepsilon)$-approximate solution for $\mathsf{MinDisAgree}[k]$. Since OPT is small, this implies that the optimal value for $\mathsf{MaxAgree}[k]$ is large which means that for any random vertex v in graph $G = (V, E)$, a lot of edges incident on v agree to the optimal clustering. Suppose we are given a random subset $S \subseteq V$ of vertices that are optimally clustered $S = \{S_1, \ldots, S_k\}$, and let us assume that S is sufficiently large. Since most of the edges in E are in agreement with the optimal clustering, we would be able to assign vertices in $V \setminus S$ to their respective clusters greedily. For any arbitrary $v \in V \setminus S$, assign v to S_i for which the number of edges that agree is maximized. Giotis and Guruswami observed that clustering vertices in $V \setminus S$ in this manner would work with high probability when these vertices belong to large clusters. For vertices in small clusters, we may not be able to decide assignments with high probability. They carry out this greedy assignment of vertices in $V \setminus S$ into clusters $S_1, \ldots, S_k$, and filter out clusters that are sufficiently large and recursively run the same procedure on the union of small clusters.

For any randomly sampled subset $S \subseteq V$ of vertices, Giotis and Guruswami try out all possible ways of partitioning S into k clusters in order to partition S optimally into k clusters $S_1, \ldots, S_k$. This ensures that at least one of the partitions matches the optimal partition. However, this exhaustive way of partitioning imposes huge burden on the running time of the algorithm. In fact, their algorithm runs in $n^{O(9^k/\varepsilon^2)} \log n$ time. Using access to the same-cluster query oracle, we can obtain a significant reduction in the running time of the algorithm. We query the oracle with pairs of vertices in S and since query answers are assumed to be consistent with some unique optimal solution, optimal k clustering of vertices in S is accomplished using at most $k|S|$ same-cluster queries. Once we have a k-partitioning of sample S that is consistent with the optimal k-clusters, we follow the remaining steps of [12]. The same approximation analysis as in [12] follows for the query algorithm. For completeness we give the modified algorithm in Fig. 3.1. Let oracle $\mathcal{A}$ take any two vertices u and v as input and return 'Yes' if they belong to the same cluster in optimal clustering, and 'No' otherwise.

Here is the main theorem giving approximation guarantee of the above algorithm. As stated earlier, the proof follows from the proof of a similar theorem (Theorem 4.7 in [12]) by Giotis and Guruswami.

Theorem 8. *Let* $0 < \varepsilon \leq 1/2$. *For any input labelling,* $\mathsf{QueryMinDisAgree}(\mathsf{k}, \varepsilon/4)$ *returns a k-clustering with the number of disagreements within a factor of* $(1+\varepsilon)$ *of the optimal.*[2]

[2] Readers familiar with [12] will realise that the statement of the theorem is slightly different from statement of the similar theorem (Theorem 13) in [12]. More specifically, the claim is about the function call with $\varepsilon/4$ as a parameter rather than ε. This is done to allow the recursive call in step (9) to be made with same value of precision parameter as the initial call. This does not change the approximation analysis but is crucial for our running time analysis.

```
QueryMinDisAgree(k, $\alpha$)
Input: Labeling $\mathcal{L} : \binom{n}{2} \rightarrow \{+,-\}$ of edges of graph $G(V, E)$, Oracle $\mathcal{A} : \binom{n}{2} \rightarrow \{\text{Yes,No}\}$.
Output: A $k$-clustering of $G$
Constants: $c_1 = \frac{1}{20}$

(1) If $k = 1$, return 1-clustering.
(2) Run QueryMaxAg on input $\mathcal{L}$ with accuracy $\frac{\alpha^2 c_1^2}{32k^4}$ to obtain $k$-clustering $ClusMax$.
(3) Set $\beta = \frac{c_1 \alpha}{16k^2}$. Pick sample $S \subseteq V$ of size $\frac{5 \log n}{\beta^2}$ independently and uniformly at random with replacement from $V$.
(4) Optimally cluster $S = \{S_1, \ldots, S_k\}$ by making same-cluster queries to oracle $\mathcal{A}$.
(5) Let $C_j = S_j$ for $1 \leq j \leq k$.
(6) For each $u \in V \setminus S$
    (6.1) $\forall i = 1, \ldots, k$: Let $l_i^u$ be the number of edges which agree between $u$ and nodes in $S_i$.
    (6.2) Let $j_u = \arg\max_i l_i^u$ be the index of the cluster which maximizes the above quantity.
    (6.3) $C_{j_u} = C_{j_u} \cup \{u\}$.
(7) Let the set of large and small clusters be $Large = \{j : 1 \leq j \leq k, |S_j| \geq \frac{n}{2k}\}$ and $Small = [k] \setminus \text{Large}$.
(8) Let $l = |Large|$, and $s = k - l$.
(9) Cluster $W = \cup_{j \in \text{Small}} S_j$ into $s$ clusters $\{W_1, \ldots, W_s\}$ using recursive calls to QueryMinDisAgree(s, $\alpha$).
(10) Let $ClusMin$ be clustering obtained by $k$ clusters $\{S_j\}_{j \in \text{Large}}$ and $\{W_t\}_{\{1 \leq t \leq s\}}$.
(11) Return the better of $ClusMin$ and $ClusMax$
```

Algorithm 3.1. Query version of the algorithm by Giotis and Guruswami.

Even though the approximation analysis of the query algorithm remains the same as the non-query algorithm of Giotis and Guruswami, the running time analysis changes significantly. Let us write a recurrence relation for the running time of our recursive algorithm. Let $T(k)$ denote the running time of the algorithm when n node graph is supposed to be clustered into k clusters with a given precision parameter α. Using the results of the previous subsection, the running time of step (2) is $O(\frac{nk^{13} \log k}{\alpha^6})$. The running time for partitioning the set S is given by $k|S|$ which is $O(\frac{k^5 \log n}{\alpha^2})$. Steps (6–8) would cost $O(nk|S|)$ time which is $O(\frac{nk^5 \log n}{\alpha^2})$. So, the recurrence relation for the running time may be written as $T(k) = T(k-1) + O(\frac{nk^{13} \log k \log n}{\alpha^6})$. This simplifies to $O(\frac{nk^{14} \log k \log n}{\alpha^6})$. As far as the same-cluster queries are concerned, we can write a similar recurrence relation. $Q(k) = Q(k-1) + O(\frac{k^{13} \log k \log n}{\alpha^6})$ which simplifies to $O(\frac{k^{14} \log k \log n}{\alpha^6})$. This completes the Proof of Theorem 2.

4 Conclusion and Open Problems

We study upper and lower bounds on the query complexity of an efficient $(1+\varepsilon)$-approximation for correlation clustering in the SSAC framework of Ashtiani et al.

An interesting open problem is to design algorithms given faulty oracles, where the query oracle gives wrong answers to queries with some probability. This setting is more practical because in some contexts it may not be known whether any two vertices belong to the same optimal cluster with high confidence. Designing an efficient $(1+\varepsilon)$-approximation algorithm for $\mathsf{MinDisAgree}[k]$ with faulty oracle is an interesting open problem.

References

1. Ailon, N., Bhattacharya, A., Jaiswal, R., Kumar, A.: Approximate clustering with same-cluster queries (2017). CoRR, abs/1704.01862. To Appear in ITCS 2018
2. Angelidakis, H., Makarychev, K., Makarychev, Y.: Algorithms for stable and perturbation-resilient problems. In: STOC, pp. 438–451 (2017)
3. Ashtiani, H., Kushagra, S., Ben-David, S.: Clustering with same-cluster queries. In: NIPS, pp. 3216–3224 (2016)
4. Awasthi, P., Balcan, M.-F, Voevodski, K.: Local algorithms for interactive clustering. In: ICML, pp. 550–558 (2014)
5. Balcan, M.-F., Blum, A.: Clustering with interactive feedback. In: Freund, Y., Györfi, L., Turán, G., Zeugmann, T. (eds.) ALT 2008. LNCS (LNAI), vol. 5254, pp. 316–328. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87987-9_27
6. Balcan, M.-F., Blum, A., Gupta, A.: Clustering under approximation stability. J. ACM (JACM) **60**(2), 8 (2013)
7. Balcan, M.F., Liang, Y.: Clustering under perturbation resilience. SIAM J. Comput. **45**(1), 102–155 (2016)
8. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. **56**(1–3), 89–113 (2004)
9. Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. J. Comput. Syst. Sci. **71**(3), 360–383 (2005)
10. Dinur, I.: The PCP theorem by gap amplification. J. ACM **54**(3), 12 (2007)
11. Fomin, F.V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y.: Tight bounds for parameterized complexity of cluster editing with a small number of clusters. J. Comput. Syst. Sci. **80**(7), 1430–1447 (2014)
12. Giotis, I., Guruswami, V.: Correlation clustering with a fixed number of clusters. In: SODA, pp. 1167–1176 (2006)
13. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001)
14. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
15. Makarychev, K., Makarychev, Y., Vijayaraghavan, A.: Correlation clustering with noisy partial information. In: COLT, pp. 1321–1342 (2015)
16. Manurangsi, P.: Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph. CoRR, abs/1611.05991 (2016)
17. Mathieu, C., Schudy, W.: Correlation clustering with noisy input. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 712–728 (2010)
18. Mazumdar, A., Saha, B.: Query complexity of clustering with side information. arXiv preprint arXiv:1706.07719 (2017)
19. Voevodski, K., Balcan, M.-F., Röglin, H., Teng, S.-H., Xia, Y.: Efficient clustering with limited distance information. In: Conference on Uncertainty in Artificial Intelligence, pp. 632–640 (2010)

Finding Tight Hamilton Cycles in Random Hypergraphs Faster

Peter Allen[1], Christoph Koch[2], Olaf Parczyk[3], and Yury Person[3](✉)

[1] Department of Mathematics, London School of Economics, Houghton Street, London WC2A 2AE, UK
p.d.allen@lse.ac.uk

[2] Mathematics Institute, University of Warwick, Zeeman Building, Coventry CV4 7AL, UK
c.koch@warwick.ac.uk

[3] Institut für Mathematik, Goethe Universität, Robert-Mayer-Str. 6-10, 60325 Frankfurt am Main, Germany
{parczyk,person}@math.uni-frankfurt.de

Abstract. In an r-uniform hypergraph on n vertices a tight Hamilton cycle consists of n edges such that there exists a cyclic ordering of the vertices where the edges correspond to consecutive segments of r vertices. We provide a first deterministic polynomial time algorithm, which finds a.a.s. tight Hamilton cycles in random r-uniform hypergraphs with edge probability at least $C \log^3 n/n$.

Our result partially answers a question of Dudek and Frieze (Random Struct Algorithms 42:374–385, 2013) who proved that tight Hamilton cycles exists already for $p = \omega(1/n)$ for $r = 3$ and $p = (e + o(1))/n$ for $r \geq 4$ using a second moment argument. Moreover our algorithm is superior to previous results of Allen et al. (Random Struct Algorithms 46:446–465, 2015) and Nenadov and Škorić (arXiv:1601.04034) in various ways: the algorithm of Allen et al. is a randomised polynomial time algorithm working for edge probabilities $p \geq n^{-1+\varepsilon}$, while the algorithm of Nenadov and Škorić is a randomised quasipolynomial time algorithm working for edge probabilities $p \geq C \log^8 n/n$.

1 Introduction

The Hamilton Cycle Problem, i.e., deciding whether a given graph contains a Hamilton cycle, is one of the 21 classical NP-complete problems due to Karp [13]. The best currently known algorithm is due to Björklund [3]: a Monte-Carlo algorithm with worst case running time $\mathcal{O}^*(1.657^n)$,[1] without false positives and false negatives occurring only with exponentially small probability. But what about "typical" instances? In other words, when the input is a random graph

The second author was supported by Austrian Science Fund (FWF): P26826, and European Research Council (ERC): No. 639046.
The third and fourth authors were supported by DFG grant PE 2299/1-1.

[1] Writing $\mathcal{O}^*$ means we ignore polylogarithmic factors.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 28–36, 2018.
https://doi.org/10.1007/978-3-319-77404-6_3

sampled from some specific distribution, is there an algorithm which finds a Hamilton cycle in polynomial time with small error probabilities?

For example, let us examine the classical binomial random graph $\mathcal{G}(n, p)$: Pósa [22] and Korshunov [15,16] proved that the hamiltonicity threshold is at $p = \Theta(\log n/n)$. Their result was improved by Komlós and Szemerédi [14] who showed that the hamiltonicity threshold coincides with the threshold for minimum degree 2, and Bollobás [4] demonstrated that this is even true for the hitting times of these two properties in the corresponding random graph process. But these results do not allow one to actually find any Hamilton cycle in polynomial time. The first polynomial time randomised algorithms for finding Hamilton cycles in $\mathcal{G}(n, p)$ are due to Angluin and Valiant [2] and Shamir [25]. Subsequently, Bollobás et al. [5] developed a deterministic algorithm, whose success probability (for input sampled from $\mathcal{G}(n, p)$) matches the probability of $\mathcal{G}(n, p)$ being hamiltonian in the limit as $n \to \infty$.

Turning to hypergraphs, there exist various notions of Hamilton cycles: weak Hamilton cycle, Berge Hamilton cycle, ℓ-overlapping Hamilton cycles (for $\ell \in [r-1]$). In each situation, one seeks to cyclically order the vertex set such that:

– any two consecutive vertices lie in a hyperedge (a *weak Hamilton cycle*),
– any two consecutive vertices lie in some chosen hyperedge and no hyperedge is chosen twice (a *Berge Hamilton cycle*),
– the edges are consecutive segments so that two consecutive edges intersect in exactly ℓ vertices (an *ℓ-overlapping Hamilton cycle*).

The (binomial) random r-uniform hypergraph $\mathcal{G}^{(r)}(n, p)$ defined on the vertex set $[n] := \{1, \ldots, n\}$, includes each *r-set* $x \in \binom{[n]}{r}$ as a *(hyper)edge* independently with probability $p = p(n)$. The study of Hamilton cycles in random hypergraphs was initiated more recently by Frieze in [10], who considered so-called loose Hamilton cycles in 3-uniform hypergraphs (these are 1-overlapping cycles in our terminology). Dudek and Frieze [7,8] determined, for all ℓ and r, the threshold for the appearance of an ℓ-overlapping Hamilton cycle in a random r-uniform hypergraph (most thresholds being determined exactly, some only asymptotically). However, these results were highly nonconstructive, relying either on a result of Johansson et al. [12] or the second moment method.

The case of weak Hamilton cycles was studied by Poole in [21], while Berge Hamilton cycles in random hypergraphs were studied by Clemens et al. in [6], the latter one being algorithmic.

In the case $\ell = r - 1$ it is customary to refer to an ℓ-overlapping cycle as a *tight* cycle. Thus, the tight r-uniform cycle on vertex set $[n]$, $n \geq r$, has edges $\{i + 1, \ldots, i + r\}$ for all i, where we identify vertex $n + i$ with i. A general result of Friedgut [9] readily shows that the threshold for the appearance of an ℓ-overlapping cycle in $\mathcal{G}^{(r)}(n, p)$ is sharp; that is, there is some threshold function $p_0 = p_0(n)$ such that for any constant $\varepsilon > 0$ the following holds. If $p \leq (1-\varepsilon)p_0$ then $\mathcal{G}^{(r)}(n, p)$ a.a.s. does not contain the desired cycle, whereas if $p \geq (1+\varepsilon)p_0$ then it a.a.s. does contain the desired cycle. Dudek and Frieze [8] proved that for $r \geq 4$ the function $p_0(n) = e/n$ is a threshold function for containment of a tight cycle, while for $r = 3$ they showed that a.a.s. $\mathcal{G}^{(3)}(n,p)$ contains a tight Hamilton cycle for any $p = p(n) = \omega(1/n)$. An easy first moment calculation shows that if $p = p(n) \le (1-\varepsilon)e/n$ then a.a.s. $\mathcal{G}^{(r)}(n,p)$ does not contain a tight Hamilton cycle.

1.1 Main Result

At the end of [8], Dudek and Frieze posed the question of finding algorithmically various ℓ-overlapping Hamilton cycles at the respective thresholds. In this paper we study tight Hamilton cycles and provide a first deterministic polynomial time algorithm, which works for p only slightly above the threshold.

Theorem 1. *For each integer $r \ge 3$ there exists $C > 0$ and a deterministic polynomial time algorithm with runtime $O(n^r)$ which for any $p \ge C(\log n)^3 n^{-1}$ a.a.s. finds a tight Hamilton cycle in the random r-uniform hypergraph $\mathcal{G}^{(r)}(n,p)$.*

Prior to our work there were two algorithms known that dealt with finding tight cycles. The first algorithmic proof was given by Allen et al. in [1], where they presented a randomised polynomial time algorithm which could find tight cycles a.a.s. at the edge probability $p \ge n^{-1+\varepsilon}$ for any fixed $\varepsilon \in (0, 1/6r)$ and running time n^{20/ε^2}. The second result is a randomised quasipolynomial time algorithm of Nenadov and Škorić [20], which works for $p \ge C(\log n)^8/n$.

Our result builds on the adaptation of the absorbing technique of Rödl et al. [24] to sparse random (hyper-)graphs. This technique was actually used earlier by Krivelevich in [17] in the context of random graphs. However, the first results that provided essentially optimal thresholds (for other problems) are proved in [1] mentioned above in the context of random hypergraphs and independently by Kühn and Osthus in [18], who studied the threshold for the appearance of powers of Hamilton cycles in random graphs. The probability of $p \ge C(\log n)^3 n^{-1}$ results in the use of so-called reservoir structures of polylogarithmic size, as first used by Montgomery to find spanning trees in random graphs [19], and later in [20]. Our improvements result in the combination of the two algorithmic approaches [1,20] and in the analysis of a simpler algorithm that we provide.

Organisation. In Sect. 2 we provide an informal overview of our algorithm. In Sect. 3 we then present two key lemmas and in Sect. 4 we give more details on the proof.

2 An Informal Algorithm Overview

2.1 Notation and Inequalities

An *s-tuple* $(u_1, \ldots, u_s)$ of vertices is an ordered set of distinct vertices. We often denote tuples by bold symbols, and occasionally also omit the brackets and write $\mathbf{u} = u_1, \ldots, u_s$. Additionally, we may also use a tuple as a set and write for example, if S is a set, $S \cup \mathbf{u} := S \cup \{u_i : i \in [s]\}$. The *reverse* of the s-tuple $\mathbf{u}$ is the s-tuple $\overleftarrow{\mathbf{s}} := (u_s, \ldots, u_1)$.

In an r-uniform hypergraph $\mathcal{G}$ the tuple $P = (u_1, \ldots, u_\ell)$ forms a *tight path* if the set $\{u_{i+1}, \ldots, u_{i+r}\}$ is an edge for every $0 \le i \le \ell - r$. For any $s \in [\ell]$ we say that P *starts* with the s-tuple $(u_1, \ldots, u_s) =: \mathbf{v}$ and *ends* with the s-tuple $(u_{\ell-(s-1)}, \ldots, u_\ell) =: \mathbf{w}$. We also call $\mathbf{v}$ the *starts-tuple* of P, $\mathbf{w}$ the *end s-tuple* of P, and P a $\mathbf{v} - \mathbf{w}$ path. The *interior* of P is formed by all its vertices but its start and end $(r-1)$-tuples. Note that the interior of P is not empty if and only if $\ell > 2(r-1)$.

For a binomially distributed random variable X and a constant $0 < \gamma < 1$ we will apply the following Chernoff-type bound (see, e.g., [11, Corollary 2.3])

$$\mathbb{P}\left[|X - \mathbb{E}(X)| \le \gamma \mathbb{E}(X)\right] \le 2 \exp\left(-\frac{\gamma^2 \mathbb{E}(X)}{3}\right). \tag{1}$$

In addition we will make use of the following consequence of Janson's inequality (see for example [11], Theorem 2.18): Let Ω be a finite set and $\mathcal{P}$ be a family of non-empty subsets of Ω. Now consider the random experiment where each $e \in \Omega$ is chosen independently with probability p and define for each $P \in \mathcal{P}$ the indicator variable I_P that each element of P gets chosen. Set $X = \sum_{P \in \mathcal{P}} I_P$ and $\Delta = \sum_{P \ne P', P \cap P' \ne \emptyset} \mathbb{E}(I_P I_{P'})$. Then

$$\mathbb{P}[X = 0] \le \exp\left(-\frac{\mathbb{E}(X)^2}{\mathbb{E}(X) + \Delta}\right). \tag{2}$$

2.2 Overview of the Algorithm

We start with the given sample of the random hypergraph $\mathcal{G}^{(r)}(n,p)$ and we will reveal the edges as we proceed. First, using the Reservoir Lemma (Lemma 1 below), we construct a tight path P_{res} which covers a small but bounded away from zero fraction of $[n]$, which has the *reservoir property*, namely that there is a set $R \subseteq V(P_{\text{res}})$ of size $2Cp^{-1} \log n \le 2n/\log^2 n$ such that for any $R' \subseteq R$, there is a tight path covering exactly the vertices $V(P_{\text{res}}) \setminus R'$ whose ends are the same as those of P_{res}, and this tight path can be found given P_{res} and R' in time polynomial in n a.a.s.

We now greedily extend P_{res}, choosing new vertices when possible and otherwise vertices in R. We claim that a.a.s. this strategy produces a structure P_{almost} which is almost a tight path extending P_{res} and covering $[n]$. The reason it is only 'almost' a tight path is that some vertices in R may be used twice. We denote the set of vertices used twice by R'_1. But we will succeed in covering $[n]$ with high probability. Recall that, due to the reservoir property, we can dispense with the vertices from R'_1 in the part P_{res} of the almost tight Hamilton path P_{almost}.

Finally, we apply the Connecting Lemma (Lemma 2 below) to find a tight path in $R \setminus R'_1$ joining the ends of P_{almost}, and using the reservoir property this gives the desired tight Hamilton cycle.

This approach is similar to that in [1]. The main difference is the way we prove the Reservoir Lemma (Lemma 1). In both [1] and this paper, we first construct many small, identical, vertex-disjoint *reservoir structures* (in some part of the literature, mostly in the dense case, this structure is called an absorber). A reservoir structure contains a spanning tight path, and a second tight path with the same ends which omits one *reservoir vertex*. We then use Lemma 2 to join the ends of all these reservoir structures together into the desired P_{res}. In [1], reservoir structures are of constant size (depending on the ε) and they are found by using brute-force search. This is slow, and is also the cause of the algorithm in [1] being randomised: there it is necessary to simulate exposure in rounds of the random hypergraph since the brute-force search reveals all edges. In this paper, by contrast, we construct reservoir structures by a local search procedure which is both much faster and reveals much less of the random hypergraph.

We will perform all the constructions in this paper by using local search procedures. At each step we reveal all the edges of $\mathcal{G}^{(r)}(n,p)$ which include a specified $(r-1)$-set, the *search base*. The number of such edges will always be in expectation of the order of pn, so that by Chernoff's inequality and the union bound, with high probability at every step in the algorithm the number of revealed edges is close to the expected number. Of course, what we may not do is attempt to reveal a given edge twice: we therefore keep track of an *exposure hypergraph* $\mathcal{E}$, which is the $(r-1)$-uniform hypergraph consisting of all the $(r-1)$-sets which have been used as search bases up to a given time in the algorithm. We will show that $\mathcal{E}$ remains quite sparse, which means that at each step we have almost as much freedom as at the start when no edges are exposed.

For concreteness, we use a doubly-linked list of vertices as the data structure representing a tight (almost-)path. However this choice of data structure is not critical to the paper and we will not further comment on it. The reader can easily verify that the various operations we describe can be implemented in the claimed time using this data structure. To simplify readability, we will omit in the calculations floor and ceiling signs whenever they are not crucial for the arguments.

3 Two Key Lemmas

Recall the definition of the *reservoir path* P_{res}. It is an r-uniform hypergraph with a special subset $R \subsetneq V(P_{\mathrm{res}})$ and some start and end $(r-1)$-tuples $\mathbf{v}$ and $\mathbf{w}$ respectively, such that:

1. P_{res} contains a tight path with the vertex set $V(P_{\mathrm{res}})$ and the 'end tuples' $\mathbf{v}$ and $\mathbf{w}$, and
2. for *any* $R' \subseteq R$, P_{res} contains a tight path with the vertex set $V(P_{\mathrm{res}})\backslash R'$ and the 'end tuples' $\mathbf{v}$ and $\mathbf{w}$.

We first give the lemma which constructs P_{res}. In addition to with high probability returning P_{res}, we also need to describe the likely resulting exposure hypergraph.

Lemma 1 (Reservoir Lemma). *For each $r \geq 3$ and $p \in (0,1]$ there exists $C > 0$ and a deterministic $O(n^r)$-time algorithm whose input is an n-vertex r-uniform hypergraph G and whose output is either 'Fail' or a reservoir path P_{res}*

with ends $\mathbf{u}$ *and* $\mathbf{v}$ *and an* $(r-1)$*-uniform exposure hypergraph* $\mathcal{E}$ *on vertex set* $V(G)$ *with the following properties.*

(i) All vertices of P_{res} *and edges of* $\mathcal{E}$ *are contained in a set* S *of size at most* $\frac{n}{4}$*.*
(ii) The reservoir $R \subseteq V(P_{\text{res}})$ *has size* $2Cp^{-1}\log n$*.*
(iii) There are no edges of $\mathcal{E}$ *contained in* $R \cup \mathbf{u} \cup \mathbf{v}$*.*
(iv) All r*-sets in* $V(G)$ *which have been exposed contain at least one edge of* $\mathcal{E}$*.*

When G *is drawn from the distribution* $\mathcal{G}^{(r)}(n,p)$ *and* $p \geq Cn^{-1}\log^3 n$*, the algorithm returns 'Fail' with probability at most* n^{-2}*.*

Furthermore we need a lemma which allows us to connect two given tuples with a not too long path. This lemma is the engine behind the proof and behind the Reservoir Lemma.

Lemma 2 (Connecting Lemma). *For each* $r \geq 3$ *there exist* $c, C > 0$ *and a deterministic* $O(n^{r-1})$*-time algorithm whose input is an* n*-vertex* r*-uniform hypergraph* G*, a pair of distinct* $(r-1)$*-tuples* $\mathbf{u}$ *and* $\mathbf{v}$*, a set* $S \subseteq V(G)$ *and an* $(r-1)$*-uniform exposure hypergraph* $\mathcal{E}$ *on the same vertex set* $V(G)$*. The output of the algorithm is either 'Fail' or a tight path of length* $o(\log n)$ *in* G *whose ends are* $\mathbf{u}$ *and* $\mathbf{v}$ *and whose interior vertices are in* S*, and an exposure hypergraph* $\mathcal{E}' \supset \mathcal{E}$*. We have that all the edges* $E(\mathcal{E}')\setminus E(\mathcal{E})$ *are contained in* $S \cup \mathbf{u} \cup \mathbf{v}$*.*

Suppose that G *is drawn from the distribution* $\mathcal{G}^{(r)}(n,p)$ *with* $p \geq C(\log n)^3/n$*, that* $\mathcal{E}$ *does not contain any edges intersecting both* S *and* $\mathbf{u} \cup \mathbf{v}$*. If furthermore* $|S| = Cp^{-1}\log n$ *and* $|e(\mathcal{E}[S])| \leq c|S|^{r-1}$ *then* $e(\mathcal{E}') \leq e(\mathcal{E}) + O(|S|^{r-2})$ *and the algorithm returns 'Fail' with probability at most* n^{-5}*.*

4 Overview Continued: More Details

We now describe the algorithm claimed by Theorem 1, which we state in a high-level overview as Algorithm 1 and explain somewhat informally some of the arguments.

Algorithm 1. Find a tight Hamilton cycle in $\mathcal{G}^{(r)}(n,p)$

```
1 use subroutine from Lemma 1 to either construct $P_{\text{res}}$ (with ends $\mathbf{u}$, $\mathbf{v}$ and
  exposure hypergraph $\mathcal{E}$ on $S$) or halt with **failure**;
  $L := V(G)\setminus S$;
  $U := S\setminus V(P_{\text{res}})$;
2 extend $P_{\text{res}}$ greedily from $v$ to cover all vertices of $U$ and using up to $n/2$
  vertices of $L$, otherwise halt with **failure**;
3 extend $P_{\text{res}}$ further greedily to $P_{\text{almost}}$ by covering all vertices of $L$ and using up
  to $|R|/2$ vertices of $R$, otherwise halt with **failure**;
4 use subroutine of Lemma 2 to connect the ends of $P_{\text{almost}}$ using the unused at
  least $|R|/2$ vertices of $R$, otherwise halt with **failure**;
```

Step 1. Given G drawn from the distribution $\mathcal{G}^{(r)}(n,p)$, we begin by applying Lemma 1 to a.a.s. find a reservoir path P_{res} with ends $\mathbf{u}$ and $\mathbf{v}$ contained in a set S of size $\frac{n}{4}$. Let $L = V(G)\backslash S$, and $U = S\backslash V(P_{\text{res}})$. Recall that by Lemma 1 (i) and (iii), all edges of $\mathcal{E}$ are contained in S; and $R \cup \mathbf{u} \cup \mathbf{v}$ contains no edges of $\mathcal{E}$. By (iv) all exposed r-sets contain an edge of $\mathcal{E}$; by choosing a little carefully where to expose edges (see Step 2 below), we will not need to worry about what exactly the edges of $\mathcal{E}$ are beyond the above information.

Step 2. We extend $P_{\text{res}} := P_0$ greedily, one vertex at a time, from its end $\mathbf{u} = \mathbf{u}_0$, to cover all of U. At each step i, we simply expose the edges of G which contain the end $\mathbf{u}_{i-1}$ of P_{i-1} and whose other vertex is not in $V(P_{i-1})$, choose one of these edges e and add the vertex from $e\backslash\mathbf{u}_{i-1}$ to P_{i-1} to form P_i. The rule we use for choosing e is the following: if i is congruent to 1 or 2 modulo 3, we choose e such that $e\backslash\mathbf{u}_{i-1}$ is in L, and if i is congruent to 0 modulo 3 we choose e such that $e\backslash\mathbf{u}_{i-1}$ is in U if it is possible; if not we choose e such that $x_i := e\backslash\mathbf{u}_{i-1}$ is in L. The point of this rule is that at each step we want to choose an edge which contains at least two vertices of L, because no such r-set can contain an edge of $\mathcal{E}$ since all the edges of $\mathcal{E}$ are contained in S (Property (i)). We will see that while $U\backslash V(P_{i-1})$ is large, we always succeed in choosing a vertex in U when i is congruent to 0 modulo 3. When it becomes small we do not, but a.a.s. we succeed often enough to cover all of U while using not more than $\frac{5n}{8}$ vertices of L.

Step 3. Next, we continue the greedy extension, this time choosing a vertex in L when possible and in R when not, until we cover all of L. It follows from the first two steps and Properties (i) and (iii) that no edge of $\mathcal{E}$ is in $L \cup R$. Thus, at each step we choose from newly exposed edges and again we a.a.s. succeed in covering L using only a few vertices of R. Let the final almost-path (which uses some vertices $R_1' \subseteq R$ twice) be P_{almost}, and R_1 the subset of R consisting of vertices we did not use in the greedy extension, i.e. $R_1 = R\backslash R_1'$.

Step 4. At last, P_{almost} covers $V(G) = L \cup U \cup V(P_{\text{res}})$. Its ends, together with the vertices of R_1, satisfy the conditions of Lemma 2, which we apply to a.a.s. complete P_{almost} to an almost-tight cycle H' in which some vertices of R_1 are used twice. The reservoir property of R now gives a tight Hamilton cycle H.

Runtime. Our applications of Lemmas 1 and 2 take time polynomial in n by the statements of those lemmas; the greedy extension procedure is trivially possible in $O(n^2)$ time (since at each extension step we just need to look at the neighbourhood of an $(r-1)$-tuple, and there are $O(n)$ steps). Finally the construction of P_{res} allows us to obtain H from H' in time $O(n^2)$: we scan through P_{res}, for each vertex r of R we scan the remainder of H' to see if it appears a second time, and if so locally reorder $V(P_{\text{res}})$ to remove r from P_{res}.

To prove Theorem 1, what remains is to justify our claims that various procedures above a.a.s. succeed.

5 Conclusion

In this paper we have improved upon the best known algorithms for finding a tight Hamilton cycle in $\mathcal{G}^{(r)}(n,p)$: we provide a deterministic algorithm with runtime $O(n^r)$ which for any edge probability $p \geq C(\log n)^3 n^{-1}$ succeeds a.a.s. While we give an affirmative answer to a question of Dudek and Frieze [8] in this regime, the question remains open for $e/n \leq p < C(\log n)^3 n^{-1}$ for $r \geq 4$, and $1/n \ll p < C(\log n)^3 n^{-1}$ for $r = 3$.

Let us now turn our attention to the closely related problem of finding the r-th power of a Hamilton cycle in the binomial random graph $\mathcal{G}(n,p)$, where $r \geq 2$. While a general result of Riordan [23] already shows that the threshold for $r \geq 3$ is given by $p = \Theta(n^{-1/r})$ (as observed in [18]), the threshold for $r = 2$ is still open, where the best known upper bound is a polylog-factor away from the first-moment lower bound $n^{-1/2}$ [20].

Since the result by Riordan is based on the second moment method it is inherently non-constructive. By contrast, the proof in [20] (for $r \geq 2$) is based on a quasi-polynomial time algorithm which for $p \geq C(\log n)^{8/r} n^{-1/r}$ finds the r-th power of an Hamilton a.a.s. in $\mathcal{G}(n,p)$, and which is very similar to their algorithm for finding tight Hamilton cycles in $\mathcal{G}^{(r)}(n,p)$. We think that our ideas are also applicable in this context and would provide an improved algorithm for finding r-th powers of Hamilton cycles in $\mathcal{G}(n,p)$, though we did not check any details.

Finally, it would be interesting to know the average case complexity of determining whether an n-vertex r-uniform hypergraph with m edges contains a tight Hamilton cycle. Our results (together with a standard link between the hypergeometric and binomial random hypergraphs) show that if $m \gg n^{r-1}\log^3 n$ then a typical such hypergraph will contain a Hamilton cycle, but the failure probability of our algorithm is not good enough to show that the average case complexity is polynomial time. For this one would need a more robust algorithm which can tolerate some 'errors' at the cost of doing extra computation to determine whether the 'error' causes hamiltonicity to fail or not.

References

1. Allen, P., Böttcher, J., Kohayakawa, Y., Person, Y.: Tight Hamilton cycles in random hypergraphs. Random Struct. Algorithms **46**(3), 446–465 (2015)
2. Angluin, D., Valiant, L.G.: Fast probabilistic algorithms for Hamiltonian circuits and matchings. J. Comput. Syst. Sci. **18**(2), 155–193 (1979)
3. Björklund, A.: Determinant sums for undirected Hamiltonicity. SIAM J. Comput. **43**(1), 280–299 (2014)
4. Bollobás, B.: The evolution of sparse graphs. In: Graph Theory and Combinatorics (Cambridge, 1983), pp. 35–57. Academic Press, London (1984). MR 777163 (86i:05119)
5. Bollobás, B., Fenner, T.I., Frieze, A.: An algorithm for finding Hamilton paths and cycles in random graphs. Combinatorica **7**(4), 327–341 (1987). MR 931191 (89h:05049)

6. Clemens, D., Ehrenmüller, J., Person, Y.: A Dirac-type theorem for Hamilton Berge cycles in random hypergraphs. Discrete Mathematical Days. Extended Abstracts of the 10th "Jornadas de matemática discreta y algorítmica" (JMDA), Barcelona, Spain, 6–8 July 2016, pp. 181–186. Elsevier, Amsterdam (2016)
7. Dudek, A., Frieze, A.: Loose Hamilton cycles in random uniform hypergraphs. Electron. J. Combin. **18**(1), Paper 48, 14 (2011). MR 2776824 (2012c:05275)
8. Dudek, A., Frieze, A.: Tight Hamilton cycles in random uniform hypergraphs. Random Struct. Algorithms **42**(3), 374–385 (2013)
9. Friedgut, E.: Sharp thresholds of graph properties, and the k-sat problem. J. Am. Math. Soc. **12**(4), 1017–1054 (1999). With an appendix by Jean Bourgain
10. Frieze, A.: Loose Hamilton cycles in random 3-uniform hypergraphs. Electron. J. Combin. **17**(1), Note 28, 4 (2010). MR 2651737 (2011g:05268)
11. Janson, S., Łuczak, T., Ruciński, A.: Random Graphs. Wiley-Interscience, New York (2000)
12. Johansson, A., Kahn, J., Vu, V.: Factors in random graphs. Random Struct. Algorithms **33**(1), 1–28 (2008). 2428975 (2009f:05243)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. IRSS, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
14. Komlós, J., Szemerédi, E.: Limit distribution for the existence of Hamiltonian cycles in a random graph. Discrete Math. **43**(1), 55–63 (1983). MR 680304 (85g:05124)
15. Korshunov, A.D.: Solution of a problem of Erdős and Renyi on Hamiltonian cycles in non-oriented graphs. Sov. Math. Dokl. **17**, 760–764 (1976)
16. Korshunov, A.D.: Solution of a problem of P. Erdős and A. Renyi on Hamiltonian cycles in undirected graphs. Metody Diskretn. Anal. **31**, 17–56 (1977)
17. Krivelevich, M.: Triangle factors in random graphs. Comb. Probab. Comput. **6**(3), 337–347 (1997)
18. Kühn, D., Osthus, D.: On Pósa's conjecture for random graphs. SIAM J. Discrete Math. **26**(3), 1440–1457 (2012)
19. Montgomery, R.: Embedding bounded degree spanning trees in random graphs. arXiv:1405.6559v2 (2014)
20. Nenadov, R., Škorić, N.: Powers of Hamilton cycles in random graphs and tight Hamilton cycles in random hypergraphs. arXiv preprint arXiv:1601.04034 (2017)
21. Poole, D.: On weak Hamiltonicity of a random hypergraph. arXiv:1410.7446 (2014)
22. Pósa, L.: Hamiltonian circuits in random graphs. Discrete Math. **14**(4), 359–364 (1976). MR 0389666 (52 #10497)
23. Riordan, O.: Spanning subgraphs of random graphs. Comb. Probab. Comput. **9**(2), 125–148 (2000)
24. Rödl, V., Ruciński, A., Szemerédi, E.: A Dirac-type theorem for 3-uniform hypergraphs. Combin. Probab. Comput. **15**(1–2), 229–251 (2006)
25. Shamir, E.: How many random edges make a graph Hamiltonian? Combinatorica **3**(1), 123–131 (1983)

Walking Through Waypoints

Saeed Akhoondian Amiri[1], Klaus-Tycho Foerster[2](✉), and Stefan Schmid[2]

[1] TU Berlin and Max Planck Institute for Informatics (Saarland), Saarbrücken, Germany
saeed.amiri@tu-berlin.de
[2] University of Vienna, Vienna, Austria
{klaus-tycho.foerster,Stefan_schmid}@univie.ac.at

Abstract. We initiate the study of a fundamental combinatorial problem: Given a capacitated graph $G = (V, E)$, find a shortest walk ("route") from a source $s \in V$ to a destination $t \in V$ that includes all vertices specified by a set $\mathscr{W} \subseteq V$: the *waypoints*. This waypoint routing problem finds immediate applications in the context of modern networked distributed systems. Our main contribution is an exact polynomial-time algorithm for graphs of bounded treewidth. We also show that if the number of waypoints is logarithmically bounded, exact polynomial-time algorithms exist even for general graphs. Our two algorithms provide an almost complete characterization of what can be solved exactly in polynomial-time: we show that more general problems (e.g., on grid graphs of maximum degree 3, with slightly more waypoints) are computationally intractable.

1 Introduction

How fast can we find a shortest route, i.e., *walk*, from a source s to a destination t which visits a given set of waypoints in a graph, but also respects edge capacities, limiting the number of traversals? This fundamental combinatorial problem finds immediate applications, e.g., in modern networked systems connecting distributed network functions. However, surprisingly little is known today about the fundamental algorithmic problems underlying *walks through waypoints*.

The problem features interesting connections to the disjoint paths problem, however, in contrast to disjoint paths, we (1) consider *walks* (of unit resource *demand* each time an edge is traversed) on *capacitated* graphs rather than *paths* on *uncapaciatated* graphs, and we (2) require that a set of specified vertices are visited. We refer to Fig. 1 for two examples.

Model. The inputs to the *Waypoint Routing Problem (WRP)* are: (1) a connected, undirected, capacitated and weighted graph $G = (V, E, c, \omega)$ consisting of $n = |V| > 1$ vertices, where $c\colon E \to \mathbb{N}$ represents edge capacities and $\omega\colon E \to \mathbb{N}$ represents the edge costs. (2) A source-destination vertex pair $s, t \subseteq V(G)$. (3) A set of k waypoints $\mathscr{W} = (w_1, \ldots, w_k) \in V(G)^k$.

https://doi.org/10.1007/978-3-319-77404-6_4

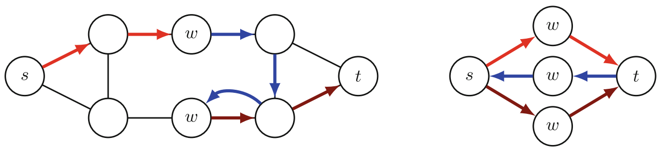


Fig. 1. Two shortest walks and their decompositions into three paths each: In both graphs, we walk through all waypoints from s to t by first taking the *red*, then the *blue*, and lastly the *brown* path. The existence of a solution in the left graph (e.g., a walk of length 7 in this case) relies on one edge incident to a waypoint having a capacity of at least two. In the right graph, it is sufficient that all edges have unit capacity. Note that no $s-t$ path through all waypoints exists, for either graph. (Color figure online)

We observe that the route (describing a walk) can be decomposed into simple paths between terminals and waypoints, and we ask: Is there a *route* R, which w.l.o.g. can be decomposed into $k+1$ path segments $R = P_1 \oplus \ldots \oplus P_{k+1}$, where:

1. *Capacities are respected:* We assume unit demands and require $|\{i \mid e \in P_i \in R, i \in [1, k+1]\}| \leq c(e)$ for every edge $e \in E$.
2. *Waypoints are visited:* Every element in $\mathscr{W}$ appears as an endpoint of exactly two distinct paths in route R and s is an endpoint of P_1 and t is an endpoint of P_{k+1}. We note that the k waypoints can be visited in any order.
3. *Walks are short:* The length $\ell = |P_1| + \ldots + |P_{k+1}|$ of route R w.r.t. edge traversal cost ω is minimal.

Remark I: Reduction to Edge-Disjoint Problems. Without loss of generality, it suffices to consider capacities $c\colon E \rightarrow \{1, 2\}$, as shown in [38, Fig. 1]: a walk R which traverses an edge e more than twice, cannot be a shortest one.

This also gives us a simple reduction of the capacitated problem to an uncapacitated (i.e., unit capacity), *edge-disjoint* problem variant, by using at most two parallel edges per original edge. Depending on the requirements, we will further subdivide these parallel edges into paths (while preserving distances and graph properties such as treewidth, at least approximately).

Remark II: Reduction to Cycles. Without loss of generality and to simplify presentation, we focus on the special case $s = t$. We show that we can modify instances with $s \neq t$ to instances with $s = t$ in a distance-preserving manner and by increasing the treewidth by at most one. Our NP-hardness results hold for $s = t$ as well. The proof is deferred to the full version of this paper.

1.1 Our Contributions

We initiate the study of a fundamental waypoint routing problem. We present polynomial-time algorithms to compute shortest routes (walks) through arbitrary waypoints on graphs of bounded treewidth and to compute shortest routes on general graphs through a bounded (but not necessarily constant) number of waypoints. We show that it is hard to significantly generalize these results both in terms of the family of graphs as well as in terms of the number of waypoints, by deriving NP-hardness results: Our exact algorithms cover a good fraction of the problem space for which polynomial-time solutions exist. More precisely, we present the following results:

1. **Shortest Walks on Arbitrary Waypoints:** While many vertex disjoint problem variants like Hamiltonian path, TSP, vertex disjoint paths, etc. are often polynomial-time solvable in graphs of bounded treewidth, their edge-disjoint counterparts (like the edge-disjoint problem), are sometimes NP-hard already on series-parallel graphs. As the Waypoint Routing Problem is an edge-based problem, one might expect that the problem is NP-hard already on bounded treewidth graphs, similarly to the edge-disjoint paths problem.
Yet, and perhaps surprisingly, we prove that a shortest walk through an arbitrary number of waypoints can be computed in polynomial time on graphs of bounded treewidth. By employing a simple trick, we transform the capacitated problem variant to an uncapacitated edge-disjoint problem: the resulting uncapacitated graph has almost the same treewidth. We then employ a well-known dynamic programming technique on a nice tree decomposition of the graph. However, since the walk is allowed to visit a vertex multiple times, we cannot rely on techniques which are known for vertex-disjoint paths. Moreover, we cannot simply use the line graph of the original graph: the resulting graph does not preserve the bounded treewidth property. Accordingly, we develop new methods and tools to deal with these issues.
2. **Shortest Walks on Arbitrary Graphs:** We show that a shortest route through a logarithmic number of waypoints can be computed in randomized time on general graphs, by reduction to the vertex-disjoint cycle problem in [7]. Similarly, we show that a route through a loglog number of waypoints can be computed in deterministic polynomial time on general graphs via [35]. Again, we show that that this is almost tight, in the sense that the problem becomes NP-hard for any polynomial number of waypoints. This reduction shows that the edge-disjoint paths problem is not harder than the vertex-disjoint problem on general graphs, and the hardness result also implies that [7] is nearly asymptotically tight in the number of waypoints.

1.2 A Practical Motivation

The problem of finding routes through waypoints or specified vertices is a natural and fundamental one. We sketch just one motivating application, arising in the context of modern networked systems. Whereas traditional computer networks were designed with an "end-to-end principle" [50] philosophy in mind, modern networks host an increasing number of "middleboxes" or network functions, distributed across the network, in order to improve performance (e.g., traffic optimizers, caches, etc.), security (e.g., firewalls, intrusion detection systems), or scalability (e.g., network address translation). Moreover, middleboxes are increasingly virtualized (a trend known as network function virtualization [23]) and can be deployed flexibly at arbitrary locations in the network (not only at the edge) and at low costs. Accordingly, also more flexible routing schemes have been developed, enabled in particular by the software-defined networking paradigm [27], to route the traffic through these (virtualized) middleboxes to compose more complex network services (also known as service chains [24]). Thus, the resulting traffic routes can be modeled as walks, and the problem of finding shortest routes through such middleboxes (the waypoints) is an instance of WRP.

1.3 Related Work

The Waypoint Routing Problem is closely related to disjoint paths problems arising in many applications [41,44,56]. Indeed, assuming unit edge capacities and a single waypoint w, the problem of finding a shortest walk (s, w, t) can be seen as a problem of finding two shortest (edge-)disjoint paths (s, w) and (w, t) with a common vertex w. More generally, a shortest walk $(s, w_1, \ldots, w_k, t)$ in a unit-capacity graph can be seen as a sequence of $k + 1$ disjoint paths. The edge-disjoint and vertex-disjoint paths problem (sometimes called *min-sum* disjoint paths) is a deep and intensively studied combinatorial problem, also in the context of parallel algorithms [36,37]. Today, we have a fairly good understanding of the *feasibility* of k-disjoint paths: for constant k, polynomial-time algorithms for general graphs have been found by Ohtsuki [45], Seymour [54], Shiloah [55], and Thomassen [57] in the 1980s, and for general k it is NP-hard [34], already on series-parallel graphs [43], i.e., graphs of treewidth at most two. However, the *optimization* problem (i.e., finding *shortest* paths) continues to puzzle researchers, even for $k = 2$. Until recently, despite the progress on polynomial-time algoritms for special graph families like variants of planar graphs [4,19,40] or graphs of bounded treewidth [51], no subexponential time algorithm was known even for the 2-disjoint paths problem on general graphs [21,29,40]. A recent breakthrough result shows that optimal solutions can at least be computed in *randomized* polynomial time [8]; however, we still have no *deterministic* polynomial-time algorithm. Both existing feasible and optimal algorithms are often impractical [8,18,52,54], and come with high time complexity. We also note that there are results on the *min-max* version of the disjoint paths problem, which asks to minimize the length of the longest path. The *min-max* problem is believed to be harder than *min-sum* [33,40].

The problem of finding shortest (edge- and vertex-disjoint) paths and cycles through k waypoints has been studied in different contexts already. The cycle problem variant is also known as the *k-Cycle Problem* and has been a central topic of graph theory since the 1960s [47]. A cycle from s through $k = 1$ waypoints back to $t = s$ can be found efficiently by breadth first search, for $k = 2$ the problem corresponds to finding a integer flow of size 2 between two vertices, and for $k = 3$, it can still be solved in linear time [30,32]; a polynomial-time solution for any constant k follows from the work on the disjoint paths problem [48]. The best known deterministic algorithm to compute *feasible* (but not necessarily shortest) paths is by Kawarabayashi [35]: it finds a cycle for up to $k = O((\log \log n)^{1/10})$ waypoints in deterministic polynomial time.

Björklund et al. [7] presented a randomized algorithm based on algebraic techniques which finds a shortest simple cycle through a given set of k vertices or edges in an n-vertex undirected graph in time $2^k n^{O(1)}$. In contrast, we assume capacitated networks and do not enforce routes to be edge or vertex disjoint, but rather consider (shortest) walks.

Regarding capacitated graphs, researchers have explored the admission control problem variant: the problem of admitting a maximal number of routing requests such that capacity constraints are met. For example, Chekuri et al. [16] and Ene et al. [22] presented approximation algorithms for maximizing the benefit of admitting disjoint paths in bounded treewidth graphs with both edge and vertex capacities. Even et al. [25,26] and Rost and Schmid [49] initiated the study of approximation algorithms for admitting a maximal number of routing *walks* through waypoints. In contrast, we focus on the optimal routing of a single walk.

In the context of capacitated graphs and single walks, the applicability of edge-disjoint paths algorithms to the so-called *ordered* Waypoint Routing problem was studied in [2,31], where the task is to find $k+1$ capacity-respecting paths $(s, w_1,), (w_1, w_2), \ldots, (w_k, t)$. An extension of their methods to the *unordered* Waypoint Routing problem via testing all possible $k!$ orderings falls short of our results: For general graphs, only $O(1)$ waypoints can be considered, and for graphs of bounded treewidth, only $O(\log n)$ waypoints can be routed in polynomial time [2]; both results concern feasibility only, but not shortest routes. We provide algorithms for $O(\log n)$ waypoints on general graphs and $O(n)$ waypoints in graphs of bounded treewidth, in both cases for shortest routes.

Lastly, for the case that all edges have a capacity of at least two and $s = t$, a direct connection of WRP to the subset traveling salesman problem (TSP) can be made [31]. In the subset TSP, the task is to find a shortest closed walk that visits a given subset of the vertices [38]. As optimal routes for WRP and subset TSP traverse every edge at most twice, optimal solutions for both are identical when $\forall e \in E : c(e) \geq 2$. Hence, we can make use of the subset TSP results of Klein and Marx, with time of $(2^{O(\sqrt{k}\log k)} + \max_{\forall e \in E} \omega(e)) \cdot n^{O(1)}$ on planar graphs. Klein and Marx also point out applicability of the dynamic programming techniques of Bellman and of Held and Karp, allowing subset TSP to be solved in time of $2^k \cdot n^{O(1)}$. For a PTAS on bounded genus graphs, we refer to [14]. We would like to note at this point that the technique for $s \neq t$ of Remark II does not apply if all edges must have a capacity of at least two. Similarly, it is in general not clear how to directly transfer $s = t$ TSP results to the case of $s \neq t$, cf. [53]. Notwithstanding, as WRP also allows for unit capacity edges (to which subset TSP is oblivious), WRP is a generalization of subset TSP.

Paper Organization. In Sect. 2 we present our results for bounded treewidth graphs and Sect. 3 considers general graphs. We derive distinct NP-hardness results in Sect. 4 and conclude in Sect. 5. Due to space constraints, some technical contents are deferred to the full version of this paper.[1]

[1] A preliminary full version is provided at [3].

2 Walking Through Waypoints on Bounded Treewidth

The complexity of the Waypoint Routing Problem on bounded treewidth graphs is of particular interest: while vertex-disjoint paths and cycles problems are often polynomial-time solvable on bounded treewidth graphs (e.g., vertex disjoint paths [48], vertex coloring, Hamiltonian cycles [6], Traveling Salesman [13], see also [11,28]) many edge-disjoint problem variants are NP-hard (e.g., edge-disjoint paths [43], edge coloring [42]). Moreover, the usual *line graph* construction approaches to transform vertex-disjoint to edge-disjoint problems are not applicable as bounded treewidth is not preserved under such transformations.

Against this backdrop, we show that indeed shortest routes through arbitrary waypoints can be computed in polynomial-time for bounded treewidth graphs.

Theorem 1. *The Waypoint Routing Problem can be solved in time of* $n^{O(tw^2)}$*, where* tw *denotes the treewidth of the network.*

In other words, the Waypoint Routing Problem is in the complexity class XP [17,20] w.r.t. treewidth. We obtain:

Corollary 1. *The Waypoint Routing Problem can be solved in polynomial time for graphs of bounded treewidth* $tw \in O(1)$*.*

Overview. We describe our algorithm in terms of a *nice tree decomposition* [39, Definition 13.1.4] (Sect. 2.1). We first transform the edge-capacitated problem into an edge-disjoint problem (on unit edge capacity graphs Sect. 2.2), leveraging a simple observation on the structure of waypoint walks and preserving distances. We show that this transformation changes the treewidth by at most an additive constant. We then define the separator signatures (Sect. 2.3) and describe how to inductively generate valid signatures in a bottom up manner on the nice tree decomposition, applying the *forget*, *join* and *introduce* operations [39, Definition 13.1.5] (Sect. 2.4).

The correctness of our approach relies on a crucial observation on the underlying Eulerian properties of the Waypoint Routing Problem in Lemma 2, allowing us to bound the number of partial walks we need to consider at the separator, see Fig. 2 for an example. Finally in Sect. 2.5, we bring together the different bits and pieces, and sketch how to dynamically program [10] the shortest waypoint walk on the rooted separator tree.

2.1 Treewidth Preliminaries

A *tree decomposition* $\mathcal{T} = (T, X)$ of a graph G consists of a bijection between a tree T and a collection X, where every element of X is a set of vertices of G such that: (1) each graph vertex is contained in at least one tree node (the *bag* or *separator*), (2) the tree nodes containing a vertex v form a connected subtree of T, and (3) vertices are adjacent in the graph only when the corresponding subtrees have a node in common.

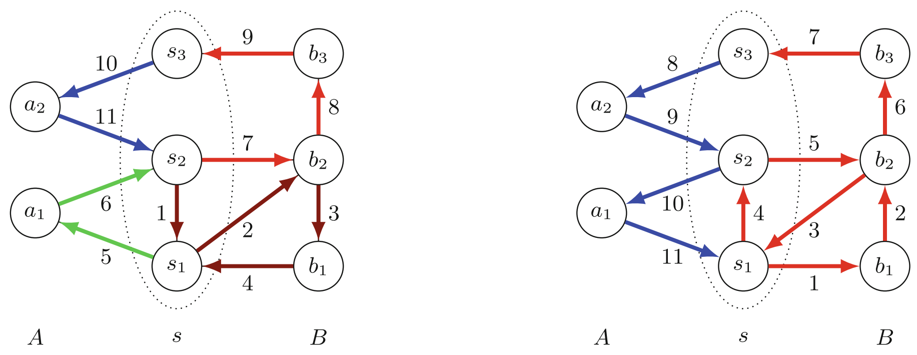


Fig. 2. Two different methods to choose an Eulerian walk, where the numbers from 1 to 11 describe the order of the traversal. In the left walk, the separator s is crossed 4 times, but only 2 times in the right walk. Furthermore, in the left walk, there are 2 walks each in $G[A]$ (green and blue) and $G[B]$ (brown and red), respectively. In the right walk, there is only 1 walk for $G[A]$ (blue) and 1 walk for $G[B]$ (red). (Color figure online)

The *width* of $\mathcal{T} = (T, X)$ is the size of the largest set in X minus 1, with the *treewidth* of G being the minimum width of all possible tree decompositions of G.

A *nice tree decomposition* is a tree decomposition such that: (1) it is rooted at some vertex r, (2) leaf nodes are mapped to bags of size 1, and (3) inner nodes are of one of three types: *forget* (a vertex leaves the bag in the parent node), *join* (two bags defined over the same vertices are merged) and *introduce* (a vertex is added to the bag in the parent node). The tree can be iteratively constructed by applying simple *forget*, *join* and *introduce* types.

Let $b \in X$ be a bag of the decomposition corresponding to a vertex $b \in V(T)$. We denote by T_b the maximal subtree of T which is rooted at bag b. By $G[b]$ we denote the subgraph of G induced on the vertices in the bag b and by $G[T_b]$ we denote the subgraph of G which is induced on vertices in all bags in $V(T_b)$. We will henceforth assume that a nice tree decomposition $\mathcal{T} = (T, X)$ of a graph G is given, covering its computation in the final steps of the proof of Theorem 1.

2.2 Unified Graphs

We begin by transforming our graphs into graphs of unit edge capacity, preserving distances and approximately preserving treewidth.

Definition 1 (Unification). *Let G be an arbitrary, edge capacitated graph. The* unified *graph G^u of G is obtained from G by the following operations on each edge $e \in E(G)$: We replace e by $c(e)$ parallel edges $e_1, \ldots, e_{c(e)}$, subdivide each resulting parallel edge by creating vertices $v_i^e, i \in [c(e)]$), and set the weight of each subdivided edge to $w(e)/2$ (i.e., the total weight is preserved). We set all edge capacities in the unified graph to 1. Similarly, given the original problem* *instance I of the Waypoint Routing Problem, the* unified instance I^u *is obtained by replacing the graph G in I with the graph G^u in I^u, without changing the waypoints, the source and the destination.*

It follows directly from the construction that I and I^u are equivalent with regards to the contained walks. Moreover, as we will see, the unification process approximately preserves the treewidth. Thus, in the following, we will focus on G^u and I^u only, and implictly assume that G and I are unified. Before we proceed further, however, let us introduce some more definitions. Using Remark I, w.l.o.g., we can focus on graphs where for all $e \in E$, $c(e) \leq 2$. The treewidth of G and G^u are preserved up to an additive constant.

Lemma 1. *Let G be an edge capacitated graph such that each edge has capacity at most 2 and let $\mathtt{tw}$ be the treewidth of G. Then G^u has treewidth at most $\mathtt{tw}+1$.*

Leveraging Eulerian Properties. A key insight is that we can leverage the Eulerian properties implied by a waypoint route. In particular, we show that the traversal of a single Eulerian walk (e.g., along an optimal solution of WRP) can be arranged s.t. it does not traverse a specified separator too often, for which we will later choose the root of the nice tree decomposition.

Lemma 2 (Eulerian Separation). *Let G be an Eulerian graph. Let s be an (A, B) separator of order $|s|$ in G. Then there is a set of $\ell \leq 2|s|$ pairwise edge-disjoint walks $\mathcal{W} = \{W_1, \dots, W_\ell\}$ of G such that*

1. *For every $W \in \mathcal{W}$, W has both of its endpoints in $A \cap B$.*
2. *Every walk $W \in \mathcal{W}$ is entirely either in $G[A]$ (as $\mathcal{W}_A$) or in $G[B]$ (as $\mathcal{W}_B$).*
3. *Let β_A be the size of the set of vertices used by $\mathcal{W}_A$ as an endpoint in s. Then, $\mathcal{W}_A$ contains at most β_A walks. Analogously, for β_B and $\mathcal{W}_B$.*
4. *There is an Eulerian walk W of G such that: $W := W_1 \oplus \ldots \oplus W_\ell$.*

2.3 Signature Generation and Properties

We next introduce the signatures we use to represent previously computed solutions to subproblems implied by the separators in the (nice) tree decomposition. For every possible signature, we will determine whether it represents a proper/ valid solution for the subproblem, and if so, store it along with an exemplary sub-solution of optimal weight.

In a nutshell, the signature describes endpoints of (partial) walks on each side of the separator. These partial walks hence need to be iteratively merged, forming signatures of longer walks through the waypoints.

Definition 2 (Signature). *Let $b \in X$. A signature σ of b (σ_b) is a pair, either containing*

1. (1) *an unordered tuple of pairs of vertices $s_i, r_i \in b$ and* (2) *a subset $E_b \subseteq E(G[b])$ with $\sigma_b = (((s_1, r_1), (s_2, r_2), \dots, (s_\ell, r_\ell)), E_b)$ s.t. $\ell \leq |b|$, or*
2. (1) $\emptyset$ *and* (2) $\emptyset$*, with $\sigma_b = (\emptyset, \emptyset)$, also called an* empty *signature $\sigma_{b,\emptyset}$.*

Note that in the above definition we may have $s_i = r_i$ for some i. We can now define a valid signature and a sub-solution, where we consider the vertex $s = t$ to be a waypoint.

Definition 3 (Valid Signature and Sub-Solution). *Let $b \in X$ and let either $\sigma_b = (\{(s_1, r_1), (s_2, r_2), \ldots, (s_\ell, r_\ell)\}, E_b)$ or $\sigma_b = \sigma_{b,\emptyset}$ be a signature of b. $\sigma_b \neq \sigma_{b,\emptyset}$ is called a* valid signature *if there is a set of pairwise edge-disjoint walks $\mathcal{W}_{\sigma_b} = \{W_1, \ldots, W_\ell\}$ such that:*

1. *If W_i is an open walk then it has both of its endpoints on (s_i, r_i), otherwise, $s_i = r_i$ and $s_i \in V(W_i)$.*
2. *Let β be the size of the set of endpoints used by σ_b. Then, it holds that $\beta \geq \ell$.*
3. *For every waypoint $w \in V(T_b)$ it holds that w is contained in some walk $W_j, 1 \leq j \leq \ell$.*
4. *Every (pairwise edge-disjoint) walk $W_j \in \mathcal{W}_{\sigma_b}$ only uses vertices from $V(T_b)$ and only edges from $E(T_b) \setminus \overline{E}_b$, with $\overline{E}_b = E(b) \setminus E_b$.*
5. *Every edge $e \in E_b$ is used by a walk in $\mathcal{W}_{\sigma_b}$.*
6. *Among all such sets of ℓ walks, $\mathcal{W}_{\sigma_b}$ has minimum total weight.*

Additionally, if for a signature $\sigma_b \neq \sigma_{b,\emptyset}$ there is such a set $\mathcal{W}_{\sigma_b}$ (possibly abbreviated by $\mathcal{W}_b$ if clear in the context), we say $\mathcal{W}$ is a valid sub-solution *in $G[T_b]$. For some waypoint contained in $G[T_b]$, we call a signature $\sigma_{b,\emptyset}$* valid, *if there is one walk W associated with it, s.t. W traverses all waypoints in $G[T_b]$, does not traverse any vertex in $V(b)$, and among all such walks in $G[T_b]$ has minimum weight. If $G[T_b]$ does not contain any waypoint, we call a signature $\sigma_{b,\emptyset}$ valid, if there is no walk associated with it.*

Lemma 3 (Number of different signatures). *There are $2^{O(|b|^2)}$ different signatures for $b \in X$.*

2.4 Programming the Nice Tree Decomposition

The nice tree decomposition directly gives us a constructive way to dynamically program WRP in a bottom-up manner. We first cover leaf nodes in Lemma 4, and then work our way up via forget (Lemma 5), introduce (Lemma 6), and join (Lemma 7) nodes, until eventually the root node is reached. Along the way, we inductively generate all valid signatures at every node.

Lemma 4 (Leaf nodes). *Let b be a leaf node in the nice tree decomposition $\mathcal{T} = (T, X)$. Then, in time $O(1)$ we can find all the valid signatures of b.*

Proof. We simply enumerate all possible valid signatures. As a leaf node only contains one vertex v from the graph, all possible edge sets in the signatures are empty, and we have two options for the pairs: First, none, second, $((v, v))$. The second option is always valid, but the first (empty) one is only valid when v is not a waypoint. □

Due to space constraints, we cannot provide the longer proofs of the other three nodes types, especially for the introduce and join nodes. Nonetheless, we at least sketch the proof idea for the join nodes, as a reduction in their time complexity would be interesting for future work, see also our remarks in Sect. 5.

Lemma 5 (Forget nodes). *Let b be a forget node in the nice tree decomposition $\mathcal{T} = (T, X)$, with one child $q = child(b)$, where we have all valid signatures for q. Then, in time $2^{O(|b|^2)}$ we can find all the valid signatures of b.*

Lemma 6 (Introduce nodes). *Let b be an introduce node in the nice tree decomposition $\mathcal{T} = (T, X)$, with one child $q = child(b)$, where we have all valid signatures for q. Then, in time $|b|^{O(|b|^2)}$ we can find all the valid signatures of b.*

Lemma 7 (Join nodes). *Let b be a join node in the nice tree decomposition $\mathcal{T} = (T, X)$, with the two children $q_1 = child(b)$ and $q_2 = child(b)$, where we have all valid signatures for q_1 and q_2. Then, in time $n^{O(|b|)} \cdot 2^{O(|b|^2)}$ we can find all the valid signatures of b.*

Our proof for join nodes consists of two parts, making use of the following fact: For a given valid signature of b, two valid sub-solutions with different path traversals have the same total length, if the set of traversed edges is identical. As thus, when trying to re-create a signature of b with a valid sub-solution, we do not need to create this specific sub-solution, but just *any* sub-solution using the *same* set of endpoints and edges. We show:

1. We can partition the edges of a valid sub-solution into two parts along a separator, resulting in a valid signature for each of the two parts, where each sub-solution uses exactly the edges in its part.
2. Given a sub-solution for each of the two parts separated, we can merge their edge sets, and create all possible signatures and sub-solutions using this merged edge set.

2.5 Putting It All Together

We now have all the necessary tools to prove Theorem 1:

Proof (Theorem 1). **Dynamically programming a nice tree decomposition.** Translating an instance of the Waypoint Routing Problem to an equivalent one with $s = t$ and unit edge capacities only increases the treewidth by a constant amount, see Remark II and Lemma 1. Although it is NP-complete to determine the treewidth of a graph and compute an according tree decomposition, there are efficient algorithms for constant treewidth [12,47]. Furthermore, Bodlaender et al. [9] presented a constant-factor approximation in a time of $O(c^{\mathtt{tw}} n)$ for some $c \in \mathbb{N}$, also beyond constant treewidth: Using their algorithm $O(\log \mathtt{tw})$ times (via binary search over the unknown treewidth size), we obtain a tree decomposition of width $O(\mathtt{tw})$. Following [39], we generate a nice tree decomposition of treewidth

$O(\texttt{tw})$ with $O(\texttt{tw}n) \in O(n^2)$ nodes in an additional time of $O(\texttt{tw}^2 n) \in O(n^3)$. The total time so far is $O(c^{\texttt{tw}} n \log \texttt{tw}) + O(\texttt{tw}^2 n)$ for some $c \in \mathbb{N}$.

We can now dynamically program the Waypoint Routing Problem on the nice tree decomposition in a bottom-up manner, using Lemma 4 (leaf nodes), Lemma 5 (forget nodes), Lemma 6 (introduce nodes), and Lemma 7 (join nodes). The time for each programming of a node is at most $O(\texttt{tw})^{O(\texttt{tw}^2)}$ or $n^{O(\texttt{tw})} \cdot 2^{O(\texttt{tw}^2)}$, meaning that we obtain all valid signatures with valid sub-solutions at the root node r, in a combined time of $n^{O(\texttt{tw}^2)}$, specifically:

$$\left(O(\texttt{tw})^{O(\texttt{tw}^2)} + n^{O(\texttt{tw})} \cdot 2^{O(\texttt{tw}^2)}\right) \cdot O(\texttt{tw} \cdot n) + O(c^{\texttt{tw}} n \log \texttt{tw}) + O(\texttt{tw}^2 n).$$

Obtaining an optimal solution. If an optimal solution I to the Waypoint Routing Problem exists (on the unified graph with $s = t$), then the traversed edges E^* and vertices V^* in I yield an Eulerian graph $G^* = (V^*, E^*)$. With each bag in the nice tree decomposition having $O(\texttt{tw})$ vertices, we can now apply (the Eulerian separation) Lemma 2: There must be a valid signature of the root r whose sub-solution uses exactly the edges E^*. As thus, from all the valid sub-solutions at r, we pick any solution to WRP with minimum weight, obtaining an optimal solution to the Waypoint Routing Problem.

3 Walking Through Logarithmically Many Waypoints

While the Waypoint Routing Problem is generally NP-hard (as we will see below), we show that a shortest walk through a bounded (not necessarily constant) number of waypoints can be computed in polynomial time. We make use of reductions to shortest vertex-disjoint [7,35] cycles problems, where the cycle has to pass through specified vertices.

Theorem 2. *For a general graph G with polynomial edge weights, a shortest walk through $k \in O(\log n)$ waypoints can be computed in randomized polynomial time, namely $2^k n^{O(1)}$. Furthermore, a walk through $k \in O\left((\log \log n)^{1/10}\right)$ waypoints in G can be computed in deterministic polynomial time.*

4 NP-Hardness

Given our polynomial-time algorithms to compute shortest walks through arbitrary waypoints on bounded treewidth graphs as well as to compute shortest walks on arbitrary graphs through a bounded number of waypoints, one may wonder whether exact polynomial time solutions also exist for more general settings. In the following, we show that this is not the case: in both dimensions (number of waypoints and more general graph families), we inherently hit computational complexity bounds. Our hardness results follow by reduction from a special subclass of NP-hard Hamiltonian cycle problems [1,15]:

Theorem 3. *WRP is NP-hard for any graph family of degree at most 3, for which the Hamiltonian Cycle problem is NP-hard.*

We have the following implication for grid graphs [5,15,46] of maximum degree 3, and can use similar ideas for the class of 3-regular bipartite planar graphs.

Corollary 2. *For any fixed constant $r \geq 1$ it holds that WRP is NP-hard on* (1) 3*-regular bipartite planar graphs and* (2) *grid graphs of maximum degree* 3*, respectively, already for $k \in O(n^{1/r})$.*

Our proof techniques also apply to the k-Cycle problem studied by Björklund et al. [7], whose solution is polynomial for logarithmic k. All possible edge-disjoint solutions are also vertex-disjoint, due to the restriction of maximum degree 3.

Corollary 3. *For any fixed constant $r \geq 1$ it holds that the k-Cycle problem is NP-hard on* (1) 3*-regular bipartite planar graphs and* (2) *grid graphs of maximum degree* 3*, respectively, already for $k \in O(n^{1/r})$.*

5 Conclusion

Motivated by the more general routing models introduced in modern software-defined and function virtualized distributed systems, we initiated the algorithmic study of computing shortest walks through waypoints on capacitated networks. We have shown, perhaps surprisingly, that polynomial-time algorithms exist for a wide range of problem variants, and in particular for bounded treewidth graphs.

In our dynamic programming approach to the Waypoint Routing Problem, parametrized by treewidth, we provided fixed-parameter tractable (FPT) algorithms for leaf, forget, and introduce nodes, but an XP algorithm for join nodes. In fact, while we do not know whether our problem can be expressed in monadic second-order logic MSO2, we can show that simply concatenating child-walks for join nodes does not result in all valid parent signatures.

We believe that our paper opens an interesting area for future research. In particular, it will be interesting to further chart the complexity landscape of the Waypoint Routing Problem, narrowing the gap between problems for which exact polynomial-time solutions do and do not exist. Moreover, it would be interesting to derive a lower bound on the runtime of (deterministic and randomized) algorithms on bounded treewidth graphs.

Acknowledgments. The authors would like to thank Riko Jacob for helpful discussions and feedback. Klaus-Tycho Foerster's and Stefan Schmid's research was partly supported by the Villum project ReNet and by Aalborg University's PreLytics project. Saeed Amiri's research was partly supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648527).

References

1. Akiyama, T., Nishizeki, T., Saito, N.: NP-completeness of the Hamiltonian cycle problem for bipartite graphs. J. Inf. Process. **3**(2), 73–76 (1980)
2. Amiri, S.A., Foerster, K.-T., Jacob, R., Schmid, S.: Charting the complexity landscape of waypoint routing. arXiv preprint arXiv:1705.00055 (2017)
3. Amiri, S.A., Foerster, K.-T., Schmid, S.: Walking through waypoints. arXiv preprint arXiv:1708.09827 (2017)
4. Akhoondian Amiri, S., Golshani, A., Kreutzer, S., Siebertz, S.: Vertex disjoint paths in upward planar graphs. In: Hirsch, E.A., Kuznetsov, S.O., Pin, J.É., Vereshchagin, N.K. (eds.) CSR 2014. LNCS, vol. 8476, pp. 52–64. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06686-8_5
5. Arkin, E.M., Fekete, S.P., Islam, K., Meijer, H., Mitchell, J.S.B., Rodríguez, Y.N., Polishchuk, V., Rappaport, D., Xiao, H.: Not being (super) thin or solid is hard: a study of grid hamiltonicity. Comput. Geom. **42**(6–7), 582–605 (2009)
6. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Appl. Math. **23**(1), 11–24 (1989)
7. Björklund, A., Husfeld, T., Taslaman, N.: Shortest cycle through specified elements. In: Proceedings of SODA (2012)
8. Björklund, A., Husfeldt, T.: Shortest two disjoint paths in polynomial time. In: Proceedings of ICALP (2014)
9. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: An approximation algorithm for treewidth. In: Proceedings of FOCS (2013)
10. Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: Lepistö, T., Salomaa, A. (eds.) ICALP 1988. LNCS, vol. 317, pp. 105–118. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-19488-6_110
11. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybern. **11**(1–2), 1–21 (1993)
12. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)
13. Bodlaender, H.L., Cygan, M., Kratsch, S., Nederlof, J.: Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. Inf. Comput. **243**, 86–111 (2015)
14. Borradaile, G., Demaine, E.D., Tazari, S.: Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. Algorithmica **68**(2), 287–311 (2014)
15. Buro, M.: Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. In: Marsland, T., Frank, I. (eds.) CG 2000. LNCS, vol. 2063, pp. 250–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45579-5_17
16. Chekuri, C., Khanna, S., Shepherd, F.B.: A note on multiflows and treewidth. Algorithmica **54**(3), 400–412 (2009)
17. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
18. Cygan, M., Marx, D., Pilipczuk, M., Pilipczuk, M.: The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In: Proceedings of FOCS (2013)

19. de Verdière, E.C., Schrijver, A.: Shortest vertex-disjoint two-face paths in planar graphs. ACM Trans. Algorithms (TALG) **7**(2), 19 (2011)
20. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0515-9
21. Eilam-Tzoreff, T.: The disjoint shortest paths problem. Discrete Appl. Math. **85**(2), 113–138 (1998)
22. Ene, A., Mnich, M., Pilipczuk, M., Risteski, A.: On routing disjoint paths in bounded treewidth graphs. In: Proceedings of SWAT (2016)
23. ETSI: Network functions virtualisation. White Paper, October 2013
24. ETSI: Network functions virtualisation (NFV); use cases. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf (2014)
25. Even, G., Medina, M., Patt-Shamir, B.: On-line path computation and function placement in SDNs. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 131–147. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49259-9_11
26. Even, G., Rost, M., Schmid, S.: An approximation algorithm for path computation and function placement in SDNs. In: Suomela, J. (ed.) SIROCCO 2016. LNCS, vol. 9988, pp. 374–390. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48314-6_24
27. Feamster, N., Rexford, J., Zegura, E.: The road to SDN. Queue **11**(12), 1–21 (2013)
28. Fellows, M., Fomin, F.V., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. In: Dress, A., Xu, Y., Zhu, B. (eds.) COCOA 2007. LNCS, vol. 4616, pp. 366–377. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73556-4_38
29. Fenner, T., Lachish, O., Popa, A.: Min-sum 2-paths problems. Theor. Comp. Sys. **58**(1), 94–110 (2016)
30. Fleischner, H., Woeginger, G.J.: Detecting cycles through three fixed vertices in a graph. Inf. Process. Lett. **42**(1), 29–33 (1992)
31. Foerster, K.-T., Parham, M., Schmid, S.: A walk in the clouds: routing through VNFs on bidirected networks. In: Proceedings of ALGOCLOUD (2017)
32. Fortune, S., Hopcroft, J.E., Wyllie, J.: The directed subgraph homeomorphism problem. Theor. Comput. Sci. **10**, 111–121 (1980)
33. Itai, A., Perl, Y., Shiloach, Y.: The complexity of finding maximum disjoint paths with length constraints. Networks **12**(3), 277–286 (1982)
34. Karp, R.M.: On the computational complexity of combinatorial problems. Networks **5**(1), 45–68 (1975)
35. Kawarabayashi, K.: An improved algorithm for finding cycles through elements. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 374–384. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68891-4_26
36. Khuller, S., Mitchell, S.G., Vazirani, V.V.: Processor efficient parallel algorithms for the two disjoint paths problem and for finding a kuratowski homeomorph. SIAM J. Comput. **21**(3), 486–506 (1992)
37. Khuller, S., Schieber, B.: Efficient parallel algorithms for testing k-connectivity and finding disjoint s-t paths in graphs. SIAM J. Comput. **20**(2), 352–375 (1991)
38. Klein, P.N., Marx, D.: A subexponential parameterized algorithm for subset TSP on planar graphs. In: Proceedings of SODA (2014)
39. Kloks, T. (ed.): Treewidth, Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0045375

40. Kobayashi, Y., Sommer, C.: On shortest disjoint paths in planar graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 293–302. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_31
41. Schrijver, A., Lovasz, L.: Paths, Flows, and VLSI-Layout. Springer-Verlag New York, Inc., Secaucus (1990). Korte, B., Promel, H.J., Graham, R.L. (eds.). ISBN 0387526854
42. Marx, D.: List edge multicoloring in graphs with few cycles. Inf. Process. Lett. **89**(2), 85–90 (2004)
43. Nishizeki, T., Vygen, J., Zhou, X.: The edge-disjoint paths problem is NP-complete for series-parallel graphs. Discrete Appl. Math. **115**, 177–186 (2001)
44. Ogier, R.G., Rutenburg, V., Shacham, N.: Distributed algorithms for computing shortest pairs of disjoint paths. IEEE Trans. Inf. Theory **39**(2), 443–455 (1993)
45. Ohtsuki, T.: The two disjoint path problem and wire routing design. In: Saito, N., Nishizeki, T. (eds.) Graph Theory and Algorithms. LNCS, vol. 108, pp. 207–216. Springer, Heidelberg (1981). https://doi.org/10.1007/3-540-10704-5_18
46. Papadimitriou, C.H., Vazirani, U.V.: On two geometric problems related to the traveling salesman problem. J. Algorithms **5**(2), 231–246 (1984)
47. Perković, L., Reed, B.A.: An improved algorithm for finding tree decompositions of small width. Int. J. Found. Comput. Sci. **11**(3), 365–371 (2000)
48. Robertson, N., Seymour, P.D.: Graph minors .XIII. The disjoint paths problem. J. Comb. Theory Ser. B **63**(1), 65–110 (1995)
49. Rost, M., Schmid, S.: Service chain and virtual network embeddings: approximations using randomized rounding. arXiv preprint arXiv:1604.02180 (2016)
50. Saltzer, J.H., Reed, D.P., Clark, D.D.: End-to-end arguments in system design. ACM Trans. Comput. Syst. **2**(4), 277–288 (1984)
51. Scheffler, P.: A practical linear time algorithm for disjoint paths in graphs with bounded tree-width. Technical report, TU Berlin (1994)
52. Schrijver, A.: Finding k disjoint paths in a directed planar graph. SIAM J. Comput. **23**(4), 780–788 (1994)
53. Sebö, A., van Zuylen, A.: The salesman's improved paths: A 3/2+1/34 approximation. In: Proceedings of FOCS (2016)
54. Seymour, D.P.: Disjoint paths in graphs. Discrete Math. **29**(3), 293–309 (1980)
55. Shiloach, Y.: A polynomial solution to the undirected two paths problem. J. ACM **27**(3), 445–456 (1980)
56. Srinivas, A., Modiano, E.: Finding minimum energy disjoint paths in wireless ad-hoc networks. Wireless Netw. **11**(4), 401–417 (2005)
57. Thomassen, C.: 2-linked graphs. Europ. J. Comb. **1**(4), 371–378 (1980)

A Collection of Lower Bounds for Online Matching on the Line

Antonios Antoniadis[2], Carsten Fischer[1], and Andreas Tönnis[3](✉)

[1] Department of Computer Science, University of Bonn, Bonn, Germany
carsten.fischer@uni-bonn.de

[2] Universität des Saarlandes and Max Plank Institut für Informatik, Saarland Campus, Saarbrücken, Germany
aantonia@mpi-inf.mpg.de

[3] Departamento de Ingeniería Matemática, Universidad de Chile, Santiago, Chile
atoennis@uni-bonn.de, atoennis@dim.uchile.cl

Abstract. In the online matching on the line problem, the task is to match a set of requests R online to a given set of servers S. The distance metric between any two points in $R \cup S$ is a line metric and the objective for the online algorithm is to minimize the sum of distances between matched server-request pairs. This problem is well-studied and – despite recent improvements – there is still a large gap between the best known lower and upper bounds: The best known deterministic algorithm for the problem is $O(\log^2 n)$-competitive, while the best known deterministic lower bound is 9.001. The lower and upper bounds for randomized algorithms are 4.5 and $O(\log n)$ respectively.

We prove that any deterministic online algorithm which in each round: (i) bases the matching decision only on information *local* to the current request, and (ii) is *symmetric* (in the sense that the decision corresponding to the mirror image of some instance I is the mirror image of the decision corresponding to instance I), must be $\Omega(\log n)$-competitive. We then extend the result by showing that it also holds when relaxing the symmetry property so that the algorithm might prefer one side over the other, but only up to some degree. This proves a barrier of $\Omega(\log n)$ on the competitive ratio for a large class of "natural" algorithms. This class includes all deterministic online algorithms found in the literature so far.

Furthermore, we show that our result can be extended to randomized algorithms that locally induce a symmetric distribution over the chosen servers. The $\Omega(\log n)$-barrier on the competitive ratio holds for this class of algorithms as well.

A. Antoniadis—Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under AN 1262/1-1.

C. Fischer—Supported by ERC Starting Grant 306465 (BeyondWorstCase).

A. Tönnis—Supported by Conicyt PCI PII 20150140. Work was done while the author was employed at the University of Bonn. Supported by ERC Starting Grant 306465 (BeyondWorstCase).

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 52–65, 2018.
https://doi.org/10.1007/978-3-319-77404-6_5

1 Introduction

The *online matching on the line problem (OML)* is a notorious special case of the *online metric matching problem (OMM)*. In both problems a set of servers $\{s_1, s_2, \ldots, s_n\} =: S$ is initially given to the algorithm, then requests from a set $R := \{r_1, r_2, \ldots, r_n\}$ arrive online one-by-one. In OMM all servers and requests are points in an arbitrary metric, while in OML all points, $r_i \in R$ and $s_j \in S$, correspond to numbers in $\mathbb{R}$ and the distance between two such points is given by the line metric, that is the 1-dimensional Euclidian metric, i.e., $d(r_i, s_j) = |r_i - s_j|$. Whenever a request r arrives it has to be matched immediately and irrevocably to an unmatched server s, and the edge $e = \langle r, s\rangle$ gets added to the matching M. The objective for the online algorithm is to minimize the sum of distances between matched server-request pairs $\min \sum_{e\in M} d(e)$.

The original motivation for OML is a scenario where items of different size have to be matched in an online fashion. For example, in a ski rental shop, skis have to be matched online to customers of approximately the same size. In this setting, customers arrive online one-by-one and have to be served immediately. In this scenario, a set of skis/customers of size x can be represented by a server/request at point x on the real line, and the length of an edge in the matching would represent the "amount of mismatch" between the corresponding pair of skis and the customer.

Since the concepts of local and local symmetric algorithms will be important for the discussion below as well as our results, we start by introducing the necessary definitions. When matching a request r to a server, one can restrict the server choice to the nearest free servers s_L and s_R, placed on the left and right of r respectively. We call these two servers the *surrounding servers* for request r. It can be easily shown by an exchange argument (see [14]), that any online algorithm can be converted to one that chooses among the surrounding servers for each request – without increasing the competitive ratio (Fig. 1).

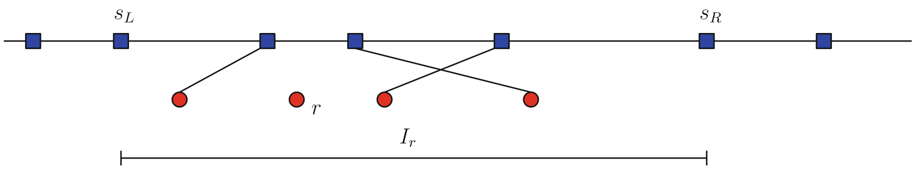


Fig. 1. We use red circles to represent requests and blue squares to represent servers. Lines show to which server each request is matched to. For a clearer representation we will often arrange servers and requests on different lines. It is without loss of generality to consider algorithms that match a request r to one of its surrounding servers s_L or s_R.

Koutsoupias and Nanavati introduced the concept of *local* algorithms which will play a central role in our results:

Definition 1 ([10]). *Let s_L and s_R be the surrounding free servers for request r. An online algorithm is called local if it serves r with one of s_L and s_R and furthermore the choice is based only upon the history of servers and requests in the* local-interval $I_r = [s_L, s_R]$ *of r.*

Note that this also implies that local algorithms are invariant with respect to parallel translation of server and request locations. Since the algorithm is only allowed to use local information, it can only use relative positions within an interval and not absolute positions of servers or requests.

One can restrict the class of local algorithms by introducing the concept of *local symmetric algorithms*. The main idea is that a local algorithm is also symmetric, if mirroring the whole interval $[s_L, s_R]$, also causes the algorithm to "mirror" its server choice.

Definition 2. *Let A be a local algorithm, and consider the arrival of a request r with a local interval $I_r = [s_L, s_R]$. Let $m = (s_L + s_R)/2$ be the point in the middle of interval I_r, and let I'_r be the reflection of I_r across point m. We say that A is* local symmetric, *if for any interval I_r when it chooses to match r to a server $s \in \{s_L, s_R\}$ then it chooses to match the mirror image of r to the mirror image of s in I'_r.*

See Fig. 2 for an example of a symmetric algorithm.

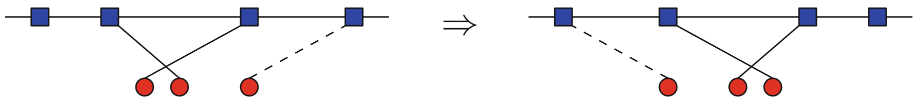


Fig. 2. The dashed line represents the choice of the algorithm. If we reflect the local-interval $[s_L, s_R]$, a local symmetric algorithm will "reflect" its decision.

A careful reader might have noticed that Definition 2 does not specify how a local symmetric algorithm behaves in the case where intervals I_r and I'_r happen to be indistinguishable[1]. However this is not important in the scope of this paper, since all of our constructions only use intervals I_r and I'_r that are clearly distinguishable from each other.

As we will see in the next subsection, a generalization of the class of local symmetric algorithms contains all studied algorithms in the literature for the problem.

1.1 Related Work

The Story so Far. For the more general OMM problem, where the underlying metric is not restricted to the line, the best possible competitive ratio is $2n -$

[1] This could be resolved by for example letting the algorithm choose the server arbitrarily or allow the adversary to force the server selection in this specific border case.

1, and algorithms that attain this ratio were analyzed in [6,8,13]. In essence all three results employ a variant of the same online *t-net-cost algorithm*. The t-net cost algorithm, for each round i, calculates a specific offline matching M_{i-1} among the server set and the $i-1$ first requests and then finds the minimum t-net cost of an augmenting path on M_{i-1} and the current request. The t-net-cost augmenting path is given by weighting the forward edges in the augmenting path by parameter t before subtracting the backward edges. This identifies a new free server to which the current request is matched. Khuller, Mitchell and Vazirani, as well as Kalyanasundaram and Pruhs [6,8] independently studied the variant of this algorithm for $t = 1$ and with M_{i-1} being the optimal offline matching on the first $i-1$ requests. They therefore augment with the classical Hungarian method, while Raghvendra [13] employs $t = n^2 + 1$ and a more complex offline matching M_{i-1}. One can easily show that all variants of the t-net cost algorithm are local symmetric when applied on the line metric.

However, in the more special case of line metrics, the linear lower bound does not hold. For a long time, the best known lower bound on the competitive ratio of the problem was 9 and it was conjectured to be tight. But in 2003, it was shown, that no deterministic algorithm for OML could be better than 9.001-competitive [4] and this remains the best known lower bound to date.

Regarding upper bounds in the line metric, Koutsoupias and Nanavati [10] studied the *work function algorithm (WFA)*, and showed that this is also $O(n)$-competitive, along with a lower bound of $\Omega(\log n)$ on its competitive ratio. Intuitively, WFA tries to balance out the 1-net-cost algorithm with the greedy algorithm that matches each request to the closest available server upon arrival. Koutsoupias and Nanavati, conjectured that WFA is $\Theta(\log n)$-competitive, but whether that is the case or not remains an open question. Although it is not straightforward that the work function algorithm is local, this was proven in [10], and symmetry easily follows by the definition of the algorithm, placing WFA in the class of local symmetric algorithms.

The first deterministic algorithm to break the linear competitive ratio was the *k-lost cows algorithm (k-LCA)* by Antoniadis et al. [1]. Their algorithm uses a connection of the matching problem to a generalization of the classical lost cow search problem from one to a larger number of cows. They give a tight analysis and prove that their algorithm is $\Theta(n^{0.58})$-competitive. It is easy to verify that the k-lost cow algorithm is local. However by design, k-LCA has a very slight "bias" towards one direction and is therefore not symmetric. A slight generalization of our construction for local symmetric algorithms is enough to capture this algorithm as well.

Very recently, Nayyar and Raghvendra [12] studied the t-net cost algorithm for a constant $t > 1$ on the line metric (actually the considered setting is slightly more general). Through a technically involved analysis they showed that for such values of t the t-net-cost algorithm is $O(\log^2 n)$-competitive. As already discussed, this algorithm is also local symmetric.

Finally, it is worth noting, and can be easily verified, that the *greedy algorithm* which matches each request to the closest free server is $\Omega(2^n)$-competitive.

Regarding randomized algorithms for the more general online metric matching problem, there are several known sub-linear algorithms. Meyerson et al. achieved a competitive ratio of $O(\log^3(n))$ for a randomized greedy algorithm on a tree embedding of the metric space [11]. Bansal et al. refined this approach and gave a $O(\log^2(n))$-competitive algorithm [2]. More recently, Gupta and Lewi gave two greedy algorithms based on tree embeddings that are $O(\log(n))$-competitive for doubling metrics – and therefore also the line metric. Additionally, they analyzed the randomized harmonic algorithm and showed that it is $O(\log(n))$-competitive for line metrics [5]. All these algorithms locally induce a symmetric distribution over the chosen servers.

An extensive survey of the OML problem can be found under [14] and [15].

Other Related Problems. A closely related problem is the transportation problem, which is a variant of online metric matching with resource augmentation. Kalyanasundaram and Pruhs [7] showed that there is a $O(1)$-competitive algorithm for this problem if the algorithm can use every server twice, whereas the offline benchmark solution only uses each server once. Subsequently, Chung et al. [3] gave a polylogarithmic algorithm for the variant where the algorithm has one additional server at every point in the metric where there is at least one server.

Another problem which resembles online matching is the k-server problem (see [9] for a survey). The main difference between the two problems is that in the k-server problem a server can be used to serve subsequent requests, while in the online matching problem a server has to be irrevocably matched to a request.

1.2 Our Contribution

Our first result is showing that any deterministic algorithm, that is local symmetric, has to be $\Omega(\log n)$-competitive.

As already mentioned the class of local symmetric algorithms includes all known algorithms except for the k-LC algorithm. However we are able to generalize the construction of our lower bound instance so that it contains an even wider class of algorithms – including k-LC. The main implication of our work is that new algorithmic insights are necessary if one hopes to obtain a $o(\log n)$-competitive algorithm for the problem.

Our lower bound instance can be seen as a full binary tree with carefully chosen distances on the edges. We describe the construction of this tree recursively. The instance is designed in such a way, that every request arrives between two subtrees, and the algorithm always has to match it to one of the two furthest leafs of these subtrees. This incurs a cost of $\Omega(n)$ at each of the $\log(n)$ levels of the tree. In contrast, the optimal solution always matches a request to neighboring server for a total cost of $O(n)$.

We complement these results with propositions that showcase the power and limitations of our construction. First, we show that any algorithm that, for every level of the lower bound instance, has a bias bounded by a factor of two with respect to the largest bias on the previous levels fulfills the conditions of our main theorem. To beat our lower bound instance an algorithm would require an asymmetric bias in its local decision routine and furthermore this bias would have to grow exponentially by more than a factor of two as the depth of the instance increases. To the best of our knowledge, the only known algorithm that features a local bias is the k-*LC* algorithm by Antoniadis et al. [1] and its bias is only $(1+\epsilon)$.

We denote that it seems hard to conceptualize a "reasonable" local algorithm for the problem that has a bias greater than two and we therefore believe/conjecture that $\Omega(\log(n))$ is likely a lower bound for the even broader class of local algorithms.

Furthermore, we show that the lower bound also applies to a wide class of randomized algorithms. If the instance can be tailored to the algorithm in such a way, that the algorithm induces a symmetric distribution over the free servers on each local subinstance, then an analogous lower bound of $\Omega(\log n)$ holds true. For this result, we have the same conditions on the bias of the algorithm as in the deterministic setting. We show that these conditions are fulfilled by the Harmonic-algorithm introduced by Gupta and Lewi [5]. This simple randomized algorithm is known to be $\Theta(\log n)$-competitive.

Due to space constraints several proofs will appear in the full version of the paper.

2 Lower Bounds for Deterministic Algorithms

This section is devoted to our main results for deterministic algorithms. First, we show that any local symmetric algorithm must be $\Omega(\log n)$-competitive. We already saw that this class of algorithms is broad and captures all known deterministic algorithms except the k-LCA. Then we extend the construction of our lower bound towards local algorithms that have a limited asymmetric bias in their decision routine.

The construction for both proofs resembles a full binary tree. It is defined recursively such that the behavior of the online algorithm on any subtree exactly mirrors its behavior on the sibling subtree. In order to achieve this, we will define for each level of the tree, intervals which consist of a new request located between two subtrees of one level lower so that the only two free servers in the interval are the ones furthest from the current request. We start at level one with simple intervals that consist of two servers and one request roughly in the center between the servers. The algorithm, by locality, will have to match the current request to one of the free servers. If, in one subtree, the algorithm matches to the right, we can create a similar subtree where the algorithm matches to the left (and vice versa) by only marginally changing the distances. We recursively repeat this construction while ensuring that, for any two sibling subtrees, the one on the left has the leftmost server free and the one on the right the rightmost one.

We start with the special case of local symmetric algorithms and give the formal construction of the lower bound instance.

At the base level, we have trees T_0 and $\overline{T}_0$ that contain a single server each. On level $i \in \{1, \ldots, k\}$, we combine two trees: T_{i-1} which has its leftmost server free, and $\overline{T}_{i-1}$ which has its rightmost server free, into an interval. We create an interval with T_{i-1} followed by request r_i at a distance 1, which is in turn followed by $\overline{T}_{i-1}$ at a distance $1+\epsilon$ to the right of r_i. If the algorithm matches r_i to the free server on the right (resp. left) in this interval then this creates tree T_i (resp. $\overline{T}_i$), and by symmetry of the algorithm if we swap the distances 1 and $1+\epsilon$ around it will match r_i to the left (resp. right) thus creating tree $\overline{T}_i$ (resp. T_i). We will see that, up to the highest tree level, T_i and $\overline{T}_i$ are always a mirror image of each other.

It is helpful to define the interval of a tree T as $I(T)$. $I(T)$ is the interval from the leftmost to the rightmost server in T just before the request of T was matched, i.e., $I(T)$ contains exactly two free servers located at its endpoints, and exactly one unmatched request contained between the two subtrees of the root of T.

Theorem 3. *Let A be a local symmetric algorithm. Then, there exists an instance with n servers such that the competitive ratio of A is in $\Omega(\log(n))$.*

Proof. We will show by induction that the interval $I(T_k)$ is always a mirror image to interval $I(\overline{T_k})$ Therefore, local symmetric online algorithms will match in one of them (w.l.o.g. in $I(T_k)$) to the right and in the other interval to the left. In this way, for each request r_i on the i-th level of the recursion, only the leftmost and rightmost servers in the respective interval are available.

Since the trees T_0 and $\overline{T}_0$ are identical, it immediately follows that intervals $I(T_1)$ and $I(\overline{T}_1)$ are mirror images of each other. Again, due to symmetry of the online algorithm, we may assume that T_1 leaves the left server open and $\overline{T}_1$ leaves its right server open.

Now, for the inductive step, intervals $I(T_i)$ and $I(\overline{T}_i)$ both take the same subtrees T_{i-1} and $\overline{T}_{i-1}$ as building blocks and those subtrees are already mirror images of each other by the inductive hypothesis. Furthermore the distances between the subtrees and request r_i are also a mirror image of one another (recall that in one tree these distances are 1 and $1+\epsilon$ and in the other one $1+\epsilon$ and 1). So, T_i and $\overline{T}_i$ must also be mirror images of each other. In addition, since in T_{i-1} the leftmost server is free and in $\overline{T}_{i-1}$ the rightmost server is free, we may adapt the construction so that T_i also leaves the leftmost server free and $\overline{T}_i$ also leaves the rightmost server free.

We have established that, when request r_i arrives, the only free servers are the left and rightmost servers s_L and s_R of $I(T_i)$ (similarly also for $I(\overline{T}_i)$). By construction, the distances are $d(r_i, s_L) \geq 2^i - 1$ and $d(r_i, s_R) \geq 2^i - 1$ because there are 2^{i-1} servers in the subtrees T_{i-1} and $\overline{T}_{i-1}$, each at a distance of $2+\epsilon$ from each other, and all of them, except the outermost are already matched. In addition, there are 2^{k-i} requests on level i in a tree of depth k. Meanwhile, the minimum distance between a request and a server is 1, so the competitive ratio is

$$\frac{c(\text{ALG})}{c(\text{OPT})} = \frac{\sum_{i=1}^{k} 2^{k-i}(2^i - 1)}{\sum_{i=1}^{k} 2^{k-i} 1} \geq \frac{k 2^k - 1}{2^k - 1} \geq k - \frac{1}{2^k} \in \Omega(k).$$

□

2.1 Local and Non-Symmetric Algorithms

We generalize the lower bound for local and symmetric algorithms that was used to prove Theorem 3. Towards this end, we define a choice function $C : I \rightarrow \{s_L, s_R\}$ that takes as input an interval I along with an unmatched request $r \in [s_L, s_R]$. The interval I is such that the only free servers it contains are s_L and s_R at the left and right end of the interval respectively. In addition, I includes information about all other matched servers and requests in between s_L and s_R. The choice function returns s_R if the algorithm decides to match r to the right and s_L otherwise. By definition, every local algorithm can be fully characterized by such a choice function.

Our construction follows a similar recursive structure starting with trees T_0 and $\overline{T}_0$ that only contain a single server. Then from level to level, we again combine two trees T_{i-1} and $\overline{T}_{i-1}$ with a request r_i in between to create a tree T_i or $\overline{T}_i$. The difference to the previous construction lies in the distances from r_i to the nearest server in the neighboring subtrees. For both trees T_i and $\overline{T}_i$ let a_i be the distance to the rightmost (and therefore closest) server of the subtree T_{i-1}. Furthermore let b_i (resp. $b_i + \epsilon$) be the distance to the left-most server of subtree T_{i-1} (resp. $\overline{T}_{i-1}$). Here, a_i and b_i are carefully chosen in such a way that $C(I(T_i)) = s_R$ and $C(I(\overline{T}_i)) = s_L$. We may assume that such a_i and b_i do always exist. If this were not the case, we could set one of them to 1, the other one to ∞, which in turn would result in an unbounded competitive ratio. Intuitively, as the distance to one of the two candidate servers grows to infinity, there must be a point at which any algorithm with a bounded competitive ratio switches to the other candidate server (Fig. 3).

Our final interval consists of two trees T_k and $\overline{T}_k$. To simplify the presentation, we skip the last request in between them. In other words, we end the process while still having two free servers. However this is without loss of generality since the adversary can present two requests, each collocated with one of the free servers, thus essentially "removing" these servers from the instance.

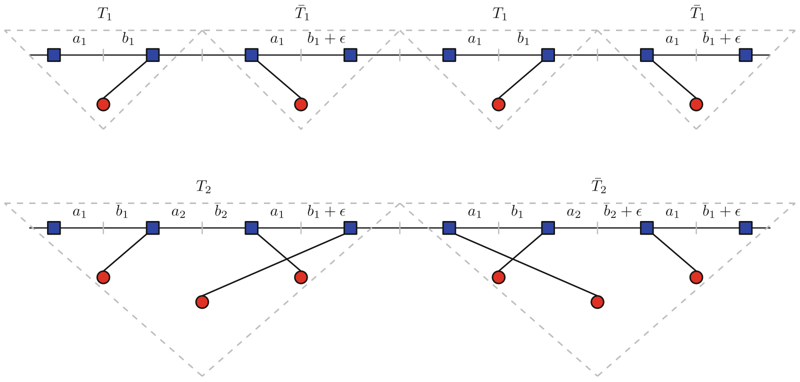


Fig. 3. The construction for the lower bound in Theorem 4.

A crucial difference to the previous proof is that although we cannot leverage symmetry of the algorithm in order to show that r_i gets matched to opposite servers in $I(T_i)$ and $I(\overline{T}_i)$, we now get this property directly by our choice of a_i's and b_i's. The analysis then follows that in the proof of Theorem 3 but is significantly more involved since the distances can now vary from level to level.

The main result of this section is the following theorem. After proving it, we discuss its implications to specific classes of algorithms.

Theorem 4. *Fix a local online algorithm A and let $x_i = a_i + b_i$, where a_i and b_i are defined for A as described above. If*

$$k - \frac{\sum_{i=1}^{k}(x_i 2^{-i} i)}{\sum_{i=1}^{k}(x_i 2^{-i})} \in \Omega(k),$$

then algorithm A is $\Omega(\log(n))$-competitive.

Proof. It can be easily shown through an exchange argument (see also [14]) that there is always an optimal solution that matches (s_i, r_i) when the servers and requests are sorted by their position. Therefore, there is an optimal solution for the instance described above that matches every request on recursion level i to the next server in the neighboring block T_{i-1} or $\overline{T}_{i-1}$. So every request on level i pays either a_i or b_i. Thus the cost of the optimal solution in the instance is the sum of the optimal solutions on T_k and $\overline{T}_k$, which differ by at most an ϵ.

$$c(\mathrm{OPT}) = 2 \cdot \min\left\{ \sum_{i=1}^{k} 2^{k-i} a_i, \frac{\epsilon}{2} + \sum_{i=1}^{k} 2^{k-i}\left(b_i + \frac{\epsilon}{2}\right)\right\} \leq \sum_{i=1}^{k} 2^{k-i}(a_i + b_i + \epsilon).$$

With ϵ arbitrarily small, its contribution to the cost is negligible. For simplicity of notation, we omit all occurrence of ϵ in the rest of the proof.

In contrast to the optimal solution, the online algorithm always matches the request r_i to the right in T_i and to the left in $\overline{T}_i$. By construction, the free server after request r_i arrived in subtree T_i is the left most server and respectively in $\overline{T}_i$ the right most server.

Thus on level i the distance to the matched server is $d(r_i, s_R) = b_i + \sum_{j=1}^{i-1} 2^{i-1-j}(a_j + b_j)$ and $d(s_L, r_i) = a_i + \sum_{j=1}^{i-1} 2^{i-1-j}(a_j + b_j)$. The instance consists of two trees T_k and $\overline{T}_k$, so the cost of the algorithms solution is

$$\begin{aligned} c(\mathrm{ALG}) &= \sum_{i=1}^{k} 2^{k-i}\left(d(s_L, r_i) + d(r_i, s_R)\right) = \sum_{i=1}^{k} 2^{k-i} \sum_{j=1}^{i} 2^{i-j}(a_j + b_j) \\ &= \sum_{i=1}^{k}\sum_{j=1}^{i} 2^{k-j}(a_j + b_j) = \sum_{i=1}^{k}(a_i + b_i)(2^{k+1-i} - 1)(k + 1 - i). \end{aligned}$$

We relabel $a_i + b_i = x_i$, this gives us

$$\frac{c(\mathrm{ALG})}{c(\mathrm{OPT})} \geq \frac{\sum_{i=1}^{k}(a_i + b_i)(2^{k+1-i} - 1)(k + 1 - i)}{\sum_{i=1}^{k}(x_i 2^{k-i})}$$
$$\geq \frac{\sum_{i=1}^{k}(x_i 2^{-i}) \cdot (k - i)}{\sum_{i=1}^{k}(x_i 2^{-i})} = k - \frac{\sum_{i=1}^{k}(x_i 2^{-i}) \cdot i}{\sum_{i=1}^{k}(x_i 2^{-i})} .$$

□

We give a sufficient condition for Theorem 4 that is easier to work with. If, for every level of the recursive construction, the bias of an online algorithm grows by at most a factor of two with respect to the maximal previous bias, then the online algorithm is $\Omega(\log n)$-competitive.

Proposition 5. *A sufficient condition for Theorem 4 is* $x_i \leq 2 \max_{j \in [1:i-1]} x_j$.

In order to show the proposition we first need the following technical lemma.

Lemma 6. *Let* $(x_i)_{i \in [1:k]}$ *be a sequence that satisfies the conditions (1)* $x_1 > 0$*; (2)* $x_i \geq 0$ *for* $i \in [1 : k]$*; and (3)* $x_i \leq 2 \max_{j \in [1:i-1]} x_j$*. Then, we have for each* $m \in [1 : k]$

$$\frac{\sum_{i=1}^{m} 2^{-i} x_i}{\sum_{i=1}^{k} 2^{-i} x_i} \geq \frac{m}{k} .$$

Proof. We will show for $m \in [1 : k - 1]$ that we have

$$\frac{1}{m} \sum_{i=1}^{m} x_i 2^{-i} \geq \frac{1}{m+1} \sum_{i=1}^{m+1} x_i 2^{-i} .$$

The statement then follows by repeated application of this identity.

Using basic calculations we can rewrite this as follows

$$\sum_{i=1}^{m} 2^{m-i} x_i \geq \frac{m}{2} \cdot x_{m+1} .$$

Since we allow x_{m+1} to be as large as $2 \max_{1 \leq i \leq m} x_i$, it satisfies to show

$$\sum_{i=1}^{m} 2^{m-i} x_i \geq m \cdot \max_{1 \leq i \leq m} x_i .$$

Let $i_1 < i_2 < \ldots < i_\ell$ denote the longest subsequence such that $x_{i_1} < x_{i_2} < \ldots < x_{i_\ell}$. Note, that $x_{i_\ell} = \max_{i \in [m]} x_i$. Furthermore, we set $i_{\ell+1} := m + 1$.

We make the following observation: Let $j < \ell - 1$. Then we have

$$\frac{2^{m-i_j} x_{i_j}}{i_{j+1} - i_j} \geq \frac{2^{m-i_{j+1}} x_{i_{j+1}}}{i_{j+2} - i_{j+1}} . \tag{1}$$

We can see this as follows: Rewriting the expression and using that $x_{i_{j+1}} \leq 2x_{i_j}$ we obtain

$$2^{i_{j+1}-i_j} \geq \frac{i_{j+1}-i_j}{i_{j+2}-i_{j+1}} \ .$$

We see that the right hand side is maximized if the denominator is equal to 1. Therefore, we obtain the estimate $2^{i_{j+1}-i_j} \geq i_{j+1}-i_j$. But this is clear, since $i_{j+1}-i_j$ is a natural number.

Then, a repeated application of (1) yields

$$\begin{aligned}\sum_{i=1}^{m} 2^{m-i} x_i &\geq \sum_{u=1}^{\ell} 2^{m-i_u} x_{i_u} \\ &= 2^{m-i_1} x_{i_1} + \sum_{u=2}^{\ell} 2^{m-i_u} x_{i_u} \\ &\geq \frac{i_2-i_1}{i_3-i_2} 2^{m-i_2} x_{i_2} + \sum_{u=2}^{\ell} 2^{m-i_u} x_{i_u} \\ &= \left(\frac{i_2-i_1}{i_3-i_2}+1\right) 2^{m-i_2} x_{i_2} + \sum_{u=3}^{\ell} 2^{m-i_u} x_{i_u} \\ &= \frac{i_3-i_1}{i_3-i_2} 2^{m-i_2} x_{i_2} + \sum_{u=3}^{\ell} 2^{m-i_u} x_{i_u} \\ &\geq \ldots \geq \frac{i_{\ell+1}-i_1}{i_{\ell+1}-i_\ell} 2^{m-i_\ell} x_{i_\ell} \ .\end{aligned}$$

At first consider the case that $i_\ell = m$. Then, the sum is lower bounded by $\frac{(m+1)-1}{(m+1)-m} 2^{m-m} x_m = m x_m$. Now assume that $i_\ell < m$. Then we want to show that

$$\frac{m+1-1}{m+1-i_\ell} 2^{m-i_\ell} \max_{i\in[m]} x_i \geq m \max_{i\in[m]} x_i \ .$$

But this is true if $2^{m-i_\ell} \geq 1+m-i_\ell$. Since $m > i_\ell$ due to our assumption, this holds true. Therefore, the statement follows. □

We are now ready to prove Proposition 5.

Proof (of Proposition 5). Now we can upper bound the expression

$$\frac{\sum_{i=1}^{k}(2^{-i}x_i)i}{\sum_{i=1}^{k} 2^{-i}x_i} \ .$$

It follows from the previous lemma that at least half of the mass of the probability distribution is located on the set $\{1,\ldots,\lceil k/2\rceil\}$. Therefore, we have

$$\frac{\sum_{i=1}^{k}(2^{-i}x_i)i}{\sum_{i=1}^{k} 2^{-i}x_i} \leq \lceil k/2\rceil/2 + k/2 \leq 3k/4 + 1/2 \ .$$

□

Furthermore, we also show that this sufficient condition is nearly tight. If the choice function has a bias that is increasing by a factor of at least $2 + \epsilon$ for some $\epsilon > 0$ with each new recursive level of the instance, then we can only give a constant lower bound on the competitive ratio. In other words, the following proposition identifies the limitations of our lower-bound instance.

Proposition 7. *If, for an online algorithm A, the corresponding instance takes the form $x_i \geq (2+\epsilon)x_{i-1}$ with $x_0 = 0$ for $\epsilon > 0$ and for all $i \in [1 : \log(n)]$, then the instance only proves a constant competitive ratio.*

3 Lower Bounds for Randomized Algorithms

Our deterministic lower bound in Sect. 2.1 also extends to randomized local algorithms. The main difference is that, in the randomized case, we cannot deduce the right distances between requests and servers a_i and b_i from a deterministic choice function. Instead, we construct the instance in such a way that the position of the free server in every subtree T_i is symmetrically distributed. Then it is easy to see that the algorithm is bound to lose at least a constant fraction in the competitive ratio over the algorithm in the deterministic case.

Again, the instance is constructed analogously to the previous section. The main difference is that now T_i consists of two subtrees T_{i-1} with an additional request r in between. Similarly to the previous subsection we set the distances between the subtrees and the new request as $d(T_{i-1}, r_i) = a_i$ and $d(r_i, T_{i-1}) = b_i$ with the exception of the top level request r_k. On level k, let $a_k = b_k = 0$. By construction, and as before, every tree T_i contains exactly one free server $s \in T_i$. For convenience of notation, $s \in [2^i]$ also denotes the position of s within T_i.

Theorem 8. *Consider any randomized online algorithm A, for which (i) a_i and b_i can be chosen in such a way that the distribution $p^i(s)$ over the position of the free server $s(T_i) \sim_{p^i} [2^i]$ is symmetric, and (ii) the distances $x_i = a_i + b_i$ fulfill*

$$k - \frac{\sum_{i=1}^{k-1} x_i 2^{-i} i}{\sum_{i=1}^{k-1} x_i 2^{-i}} \in \Omega(k) ,$$

for every $i \in [k-1]$. Algorithm A is $\Omega(\log n)$-competitive on the instance $T_{\log n}$.

Proof. If the distribution p^i over the position of the free server s in T_i is symmetric, then with probability $\frac{1}{2}$ the server is in the first half of T_i. In this case $s \leq 2^{i-1}$. Analogously, with probability $\frac{1}{2}$, we also have $s \geq 2^{i-1} + 1$. If both events occur at the same time, then in T_{i+1} the distance between matched server and request is at least $\min\{d(s_L, r_i), d(r_i, s_R)\} \geq \sum_{j=1}^{i-1}(a_j + b_j)2^{i-1-j}$. Here, we omit the cost of a_i or b_i because the algorithm will not pay both.

Therefore, the expected cost of the online algorithm is at least

$$\begin{aligned}\mathbf{E}\left[c(\text{ALG})\right] &\geq \sum_{i=1}^{k} \frac{1}{4} 2^{k-i} \sum_{j=1}^{i-1} (a_j + b_j) 2^{i-1-j} \\ &= \frac{1}{4} \sum_{i=1}^{k-1} (a_i + b_i)(2^{k-i-1} - 1)(k-i) \ .\end{aligned}$$

Similar to the previous section, the cost of the optimal solution are

$$c(\text{OPT}) = \min \left\{ \sum_{i=1}^{k} 2^{k-i} a_i, \sum_{i=1}^{k} 2^{k-i} b_i \right\} \leq \sum_{i=1}^{k} 2^{k-i-1} (a_i + b_i) \ .$$

Again we substitute $x_i = (a_i + b_i)$, then the lower bound instance guarantees a competitive ratio of at least

$$\frac{\mathbf{E}\left[c(\text{ALG})\right]}{c(\text{OPT})} \geq \frac{\frac{1}{4}\sum_{i=1}^{k-1} x_i (2^{k-i-1} - 1)(k-i)}{\sum_{i=1}^{k} 2^{k-i-1} x_i} \geq \frac{\frac{1}{8}\sum_{i=1}^{k-1} x_i 2^{-i} (k-i)}{\sum_{i=1}^{k-1} x_i 2^{-i}} \ .$$

In the last step, we use that $x_k = a_k + b_k = 0$. Now we have an expression similar to the previous proof, the same steps give the desired result. □

An example for a randomized online algorithm for OML is the Harmonic algorithm by Gupta and Lewi [5]. They have shown that this algorithm is $O(\log n)$-competitive in expectation. We show that Harmonic fulfills the condition in Theorem 8, and therefore provide an alternative that Harmonic is $\Omega(\log n)$-competitive.

Proposition 9. *For the algorithm* Harmonic*,* $a_i = b_i = 1$ *yields a symmetric distribution* $p^i(s)$ *for all* $i \in [k]$*.*

4 Discussion

This paper rules out an $o(\log(n))$-competitive ratio for a wide class of both deterministic and randomized algorithms for OML. This means that new algorithmic insights are necessary if one hopes to obtain such a $o(\log(n))$-competive algorithm for the problem. It is natural to try and conceptualize a "reasonable" deterministic/randomized algorithm that beats our instance. As already mentioned, we find it particularly hard to conceptualize such a local algorithm and we therefore conjecture that the lower bound of $\Omega(\log n)$ holds for all local algorithms, even though a different construction would be required to handle algorithms with alternating and exponentially growing bias. However, it would be interesting to try to design and analyze a non-local algorithm, that also employs information from outside of the local-interval in order to match a request.

The best deterministic algorithm known so far is $O(\log^2(n))$-competitive, and for many known algorithms the best known lower bound on their competitive ratio is $\Omega(\log(n))$, it would be reasonable to work on a tighter analysis for an existing algorithm in order to (hopefully) prove it $\Theta(\log(n))$-competitive. The WFA algorithm has been conjectured to be $\Theta(\log n)$-competitive before and our work does not change anything on that front. Another promising candidate is the t-net-cost algorithm for some $t > 1$ because this is the currently best known algorithm and it is not obvious that the analysis is tight for the line metric.

References

1. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: A o(n)-competitive deterministic algorithm for online matching on a line. In: Proceedings of 12th International Workshop Approximations and Online Algorithms (WAOA), pp. 11–22 (2014)
2. Bansal, N., Buchbinder, N., Gupta, A., Naor, J.: A randomized o(log2 k)-competitive algorithm for metric bipartite matching. Algorithmica **68**(2), 390–403 (2014)
3. Chung, C., Pruhs, K., Uthaisombut, P.: The online transportation problem: on the exponential boost of one extra server. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 228–239. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78773-0_20
4. Fuchs, B., Hochstättler, W., Kern, W.: Online matching on a line. Theo. Comput. Sci. **332**(1–3), 251–264 (2005)
5. Gupta, A., Lewi, K.: The online metric matching problem for doubling metrics. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7391, pp. 424–435. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31594-7_36
6. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
7. Kalyanasundaram, B., Pruhs, K.: The online transportation problem. SIAM J. Discrete Math. **13**(3), 370–383 (2000)
8. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theo. Comput. Sci. **127**(2), 255–267 (1994)
9. Koutsoupias, E.: The k-server problem. Comput. Sci. Rev. **3**(2), 105–118 (2009)
10. Koutsoupias, E., Nanavati, A.: The online matching problem on a line. In: Solis-Oba, R., Jansen, K. (eds.) WAOA 2003. LNCS, vol. 2909, pp. 179–191. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24592-6_14
11. Meyerson, A., Nanavati, A., Poplawski, L.J.: Randomized online algorithms for minimum metric bipartite matching. In: Proceedings of 17th Symposium Discrete Algorithms (SODA), pp. 954–959 (2006)
12. Nayyar, K., Raghvendra, S.: An input sensintive online algorithm for the metric bipartite matching problem. In: FOCS (2017, to appear). http://ieee-focs.org/FOCS-2017-Papers/3464a505.pdf
13. Raghvendra, S.: A robust and optimal online algorithm for minimum metric bipartite matching. In: Approximation, Randomization, and Combinatorial Optimization, APPROX/RANDOM, pp. 18:1–18:16 (2016)
14. van Stee, R.: SIGACT news online algorithms column 27: online matching on the line, part 1. SIGACT News **47**(1), 99–110 (2016)
15. van Stee, R.: SIGACT news online algorithms column 28: online matching on the line, part 2. SIGACT News **47**(2), 40–51 (2016)

On the Complexity of Finding Internally Vertex-Disjoint Long Directed Paths

Júlio Araújo[1], Victor A. Campos[2], Ana Karolinna Maia[2], Ignasi Sau[1,3(✉)], and Ana Silva[1]

[1] ParGO Research Group, Departamento de Matemática, Universidade Federal do Ceará, Fortaleza, Brazil
{julio,anasilva}@mat.ufc.br

[2] ParGO Research Group, Departamento de Computação, Universidade Federal do Ceará, Fortaleza, Brazil
{campos,karolmaia}@lia.ufc.br

[3] CNRS, Université de Montpellier, LIRMM, Montpellier, France
ignasi.sau@lirmm.fr

Abstract. For two positive integers k and ℓ, a $(k \times \ell)$*-spindle* is the union of k pairwise internally vertex-disjoint directed paths with ℓ arcs each between two vertices u and v. We are interested in the (parameterized) complexity of several problems consisting in deciding whether a given digraph contains a subdivision of a spindle, which generalize both the Maximum Flow and Longest Path problems. We obtain the following complexity dichotomy: for a fixed $\ell \geq 1$, finding the largest k such that an input digraph G contains a subdivision of a $(k \times \ell)$-spindle is polynomial-time solvable if $\ell \leq 3$, and NP-hard otherwise. We place special emphasis on finding spindles with exactly two paths and present FPT algorithms that are asymptotically optimal under the ETH. These algorithms are based on the technique of representative families in matroids, and use also color-coding as a subroutine. Finally, we study the case where the input graph is acyclic, and present several algorithmic and hardness results.

Keywords: Digraph subdivision · Spindle · Parameterized complexity FPT algorithm · Representative family · Complexity dichotomy

1 Introduction

A *subdivision* of a digraph F is a digraph obtained from F by replacing each arc (u, v) of F by a directed (u, v)-path. We are interested in the (parameterized) complexity of several problems consisting in deciding whether a given digraph contains as a subdigraph a subdivision of a *spindle*, defined as follows. For k positive integers $\ell_1, \ldots, \ell_k$, an $(\ell_1, \ldots, \ell_k)$-*spindle* is the digraph containing k paths $P_1, \ldots, P_k$ from a vertex u to a vertex v, such that $|E(P_i)| = \ell_i$ for

Work supported by DE-MO-GRAPH grant ANR-16-CE40-0028 and CNPq grant 306262/2014-2.

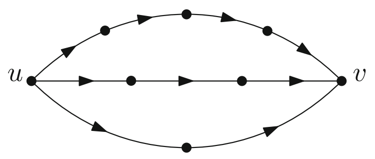


Fig. 1. A $(4, 3, 2)$-spindle. This digraph contains a subdivision of a (3×2)-spindle, but not of a (3×3)-spindle.

$1 \leq i \leq k$ and $V(P_i) \cap V(P_j) = \{u, v\}$ for $1 \leq i \neq j \leq k$. If $\ell_i = \ell$ for $1 \leq i \leq k$, an $(\ell_1, \ldots, \ell_k)$-spindle is also called a $(k \times \ell)$-*spindle*. See Fig. 1 for an example.

Note that a digraph G contains a subdivision of a $(k \times 1)$-spindle if and only if there exist two vertices u and v and k internally vertex-disjoint paths from u to v. On the other hand, G contains a subdivision of a $(1 \times \ell)$-spindle if and only if G contains a path of length at least ℓ. Hence, finding a subdivision of a spindle generalizes both the Maximum Flow and Longest Path problems.

Subdivisions of spindles were considered by Bang-Jensen et al. [4], who introduced the general problem of finding a subdivision of a *fixed* digraph F and presented NP-hardness results and polynomial-time algorithms for several choices of F (see also [13]). In particular, they proved that when F is a spindle, the problem can be solved in time $n^{O(|V(F)|)}$ by a combination of brute force and a flow algorithm. Using terminology from parameterized complexity, this means that the problem is in XP parameterized by the size of F, and they left open whether it is FPT. Note that on undirected graphs the notion of subdivision coincides with that of topological minor, hence by Grohe et al. [12] the problem is FPT parameterized by the size of F, for a general graph F. We refer to the introduction of [4] for a more detailed discussion about problems related to containment relations on graphs and digraphs.

We first consider the following two optimization problems about finding subdivisions of spindles: (1) for a fixed positive integer k, given an input digraph G, find the largest integer ℓ such that G contains a subdivision of a $(k \times \ell)$-spindle, and (2) for a fixed positive integer ℓ, given an input digraph G, find the largest integer k such that G contains a subdivision of a $(k \times \ell)$-spindle. We call these problems Max $(k \times \bullet)$-Spindle Subdivision and Max $(\bullet \times \ell)$-Spindle Subdivision, respectively. We prove that the first problem is NP-hard for any integer $k \geq 1$, by a simple reduction from Longest Path. The second problem turns out to be much more interesting, and we achieve the following dichotomy.

Theorem 1. *Let $\ell \geq 1$ be a fixed integer.* Max $(\bullet \times \ell)$-Spindle Subdivision *is polynomial-time solvable if $\ell \leq 3$, and* NP*-hard otherwise, even restricted to acyclic digraphs.*

The reduction for the NP-hard cases is inspired by a result of Brewster et al. [6] to prove the NP-hardness of packing vertex-disjoint paths on digraphs. Concerning the polynomial algorithms, to solve the case $\ell = 3$, which is the only nontrivial one, we use a *vertex splitting procedure* that builds on ideas of

Schrijver [19] on undirected graphs and by Kriesell [16] on directed graphs (see also [3, Sect. 5.9]).

It worth mentioning that both the positive and negative results of Theorem 1 hold as well for the case where the endvertices of the desired spindle are *fixed*. Itai et al. [14] considered the problems of, given a digraph G and two distinct vertices s and t, finding the maximum number of internally vertex-disjoint (s, t)-paths whose lengths are *at most* or *exactly equal* to a fixed constant ℓ, and achieved dichotomies for both cases. Note that the problem we consider corresponds to a constraint of type '*at least*' on the lengths of the desired paths. Hence, Theorem 1 together with the results of Itai et al. [14] provide a full picture of the complexity of finding a maximum number of length-constrained internally vertex-disjoint directed (s, t)-paths.

We place special emphasis on finding subdivisions of spindles with exactly two paths, which we call 2-*spindles*. The existence of subdivisions of 2-spindles has attracted some interest in the literature. Indeed, Benhocine and Wojda [5] showed that a tournament on $n \geq 7$ vertices always contains a subdivision of an (ℓ_1, ℓ_2)-spindle such that $\ell_1 + \ell_2 = n$. More recently, Cohen et al. [7] showed that a strongly connected digraph with chromatic number $\Omega((\ell_1+\ell_2)^4)$ contains a subdivision of an (ℓ_1, ℓ_2)-spindle, and this bound was subsequently improved to $\Omega((\ell_1 + \ell_2)^2)$ by Kim et al. [15], who also provided improved bounds for Hamiltonian digraphs.

We consider two problems concerning the existence of subdivisions of 2-spindles. The first one is, given an input digraph G, find the largest integer ℓ such that G contains a subdivision of an (ℓ_1, ℓ_2)-spindle with $\min\{\ell_1, \ell_2\} \geq 1$ and $\ell_1 + \ell_2 = \ell$. We call this problem Max $(\bullet, \bullet)$-Spindle Subdivision, and we show the following results.

Theorem 2. *Given a digraph G and a positive integer ℓ, the problem of deciding whether there exist two strictly positive integers ℓ_1, ℓ_2 with $\ell_1+\ell_2 = \ell$ such that G contains a subdivision of an (ℓ_1, ℓ_2)-spindle is* NP*-hard and* FPT *parameterized by ℓ. The running time of the* FPT *algorithm is $2^{O(\ell)} \cdot n^{O(1)}$, and the function $2^{O(\ell)}$ is asymptotically optimal unless the* ETH *fails. Moreover, the problem does not admit polynomial kernels unless* NP $\subseteq$ coNP/poly.

The second problem is, for a fixed strictly positive integer ℓ_1, given an input digraph G, find the largest integer ℓ_2 such that G contains a subdivision of an (ℓ_1, ℓ_2)-spindle. We call this problem Max $(\ell_1, \bullet)$-Spindle Subdivision, and we show the following results.

Theorem 3. *Given a digraph G and two integers ℓ_1, ℓ_2 with $\ell_2 \geq \ell_1 \geq 1$, the problem of deciding whether G contains a subdivision of an (ℓ_1, ℓ_2)-spindle can be solved in time $2^{O(\ell_2)} \cdot n^{O(\ell_1)}$. When ℓ_1 is a constant, the problem remains* NP*-hard and the running time of the* FPT *algorithm parameterized by ℓ_2 is asymptotically optimal unless the* ETH *fails. Moreover, the problem does not admit polynomial kernels unless* NP $\subseteq$ coNP/poly.

The hardness results of Theorems 2 and 3 are based on a simple reduction from Directed Hamiltonian Cycle. Both FPT algorithms, which are our main technical contribution, are based on the technique of *representative families* in matroids introduced by Monien [18], and in particular its improved version recently presented by Fomin et al. [10]. The FPT algorithm of Theorem 3 also uses the *color-coding* technique of Alon et al. [1] as a subroutine.

Finally, we consider the case where the input digraph G is *acyclic*. We prove the following result by using a standard dynamic programming algorithm.

Theorem 4. *Given an acyclic digraph G and two positive integers k, ℓ, the problem of deciding whether G contains a subdivision of a $(k \times \ell)$-spindle can be solved in time $O(\ell^k \cdot n^{2k+1})$.*

The above theorem implies, in particular, that when k is a constant the problem is polynomial-time solvable on acyclic digraphs, which generalizes the fact that Longest Path, which corresponds to the case $k = 1$, is polynomial-time solvable on acyclic digraphs (cf. [20]).

As observed by Bang-Jensen et al. [4], from the fact that the k-Linkage problem is in XP on acyclic digraphs [17], it easily follows that finding a subdivision of a general digraph F is in XP on DAGs parameterized by $|V(F)|$. Motivated by this, we prove two further hardness results about finding subdivisions of spindles on DAGs. Namely, we prove that if F is the disjoint union of (2×1)-spindles, then finding a subdivision of F is NP-complete on planar DAGs, and that if F is the disjoint union of a $(k_1 \times 1)$-spindle and a $(k_2 \times 1)$-spindle, then finding a subdivision of F is W[1]-hard on DAGs parameterized by $k_1 + k_2$. These two results should be compared to the fact that finding a subdivision of a single $(k \times 1)$-spindle can be solved in polynomial time on general digraphs by a flow algorithm.

Organization of the paper. In Sect. 2 we prove Theorem 1, and in Sect. 3 we prove Theorems 2 and 3. Our results about acyclic digraphs are deferred to the full version of this article [2]. In Sect. 4 we present some open problems for further research. Due to space limitations, the proofs of the results marked with '($\star$)' are deferred to the full version.

We use standard notation concerning (di)graphs, parameterized complexity, and matroids. For completeness, some basic preliminaries can be found in the full version. For a positive integer k, we denote by $[k]$ the set of all integers i such that $1 \leq i \leq k$. Throughout the article, unless stated otherwise, we let n denote the number of vertices of the input digraph of the problem under consideration.

2 Complexity Dichotomy in Terms of the Path Lengths

In this section we focus on the two natural optimization versions of finding subdivisions of spindles mentioned in the introduction, namely Max $(k \times \bullet)$-Spindle Subdivision and Max $(\bullet \times \ell)$-Spindle Subdivision.

It is easy to prove that the first problem is NP-hard for any integer $k \geq 1$, by a simple reduction from Longest Path.

Theorem 5. *Let $k \geq 1$ be a fixed integer. The* Max $(k \times \bullet)$-Spindle Subdivision *problem is* NP*-hard.*

Proof. We provide a polynomial reduction from the Longest Path problem on general digraphs, which is NP-hard as it generalizes Hamiltonian Path [11]. For $k = 1$, Max $(k \times \bullet)$-Spindle Subdivision is exactly the Longest Path problem, and the result follows. For $k > 1$, let G be an instance of Longest Path with n vertices, and we build an instance G' of Max $(k \times \bullet)$-Spindle Subdivision as follows. We start with G and we add to it $2k - 2$ new vertices $s_1, \ldots, s_{k-1}, t_1, \ldots, t_{k-1}$. For $i \in [k-1]$, we add an arc from every vertex of G to s_i, and arc from t_i to every vertex of G, and a path from s_i to t_i with n edges through $n - 1$ new vertices. This completes the construction of G'. It is clear that the length of a longest path in G equals the largest integer k such that G' contains a subdivision of a $(k \times \ell)$-spindle, concluding the proof. □

We now present the complexity dichotomy for the second problem, in order to prove Theorem 1. We start with the hardness result.

Theorem 6. *Let $\ell \geq 4$ be a fixed integer. The* Max $(\bullet \times \ell)$-Spindle Subdivision *problem is* NP*-hard, even when restricted to DAGs.*

Proof. We provide a polynomial reduction from 3-Dimensional Matching, which is NP-hard [11]. In the 3-Dimensional Matching problem, we are given three sets A, B, C of the same size and a set of triples $\mathcal{T} \subseteq A \times B \times C$. The objective is to decide whether there exists a set $\mathcal{T}' \subseteq \mathcal{T}$ of pairwise disjoint triples with $|\mathcal{T}'| = |A|$. Given an instance $(A, B, C, \mathcal{T})$ of 3-Dimensional Matching, with $|A| = n$ and $\mathcal{T} = m$, we construct an instance G of Max $(\bullet \times \ell)$-Spindle Subdivision as follows. We first present the reduction for $\ell = 4$, and then we explain how to modify it for a general $\ell > 4$.

For every $i \in [n]$, we add to G three vertices a_i, b_i, c_i, corresponding to the elements in the sets A, B, C, respectively. Let H the digraph with vertices $x_0, x_1, y_0, y_1, z_0, z_1, a, b, c$ and arcs (x_0, x_1), (x_1, a), (x_1, y_0), (y_0, y_1), (y_1, b), (x_0, z_0), (z_0, z_1), (z_1, c) (see Fig. 2(a)). For every triple $T \in \mathcal{T}$, with $T = (a_i, b_j, c_p)$, we add to G a copy of H and we identify vertex a with a_i, vertex b with b_j, and vertex c with c_p. Finally, we add a new vertex s that we connect to all other vertices introduced so far, and another vertex t to which we connect all other vertices introduced so far except s.

The constructed digraph G is easily seen to be a DAG. Indeed, we can define a topological ordering of $V(G)$ so that all arcs go from left to right as follows. We select s (resp. t) as the leftmost (resp. rightmost) vertex. We divide the remaining vertices of G into two blocks. On the right, we place all the vertices $\{a_i, b_i, c_i : i \in [n]\}$, and we order them arbitrarily. On the left, we place the remaining vertices of G, which we also order arbitrarily, except that for every triple $T \in \mathcal{T}$, we order the vertices in its copy of H, distinct from a, b, c, such that $x_0 < x_1 < y_0 < y_1 < z_0 < z_1$ holds. One can check that, with respect to this ordering, all the arcs of G go from left to right.

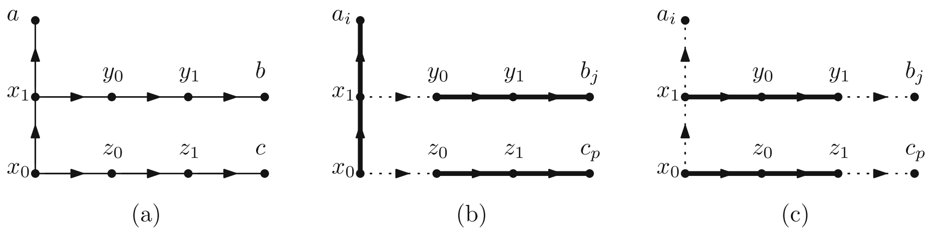


Fig. 2. (a) Digraph H. (b) Selected paths when $T \in \mathcal{T}'$. (c) Selected paths when $T \in \mathcal{T} \setminus \mathcal{T}'$.

Note that $|V(G)| = 3n + 6m + 2$, and therefore the largest integer k for which G contains a subdivision of a $(k \times 4)$-spindle is $k^* := n + 2m$, as each path involved in such a spindle contains at least three vertices distinct from its endpoints. The following claim concludes the proof for $\ell = 4$.

Claim 1. *$(A, B, C, \mathcal{T})$ is a* Yes*-instance of* 3-Dimensional Matching *if and only if G contains a subdivision of a $(k^* \times 4)$-spindle.*

Proof of the claim. Suppose first that $(A, B, C, \mathcal{T})$ is a Yes-instance, and let $\mathcal{T}' \subseteq \mathcal{T}$ be a solution. We proceed to define a set $\mathcal{P}$ of $n + 2m$ vertex-disjoint 2-paths in $G \setminus \{s, t\}$, which together with s and t yield the desired spindle. For every $T \in \mathcal{T}'$, with $T = (a_i, b_j, c_p)$, we add to $\mathcal{P}$ the three paths (x_0, x_1, a_i), (y_0, y_1, b_j), and (z_0, z_1, c_p) (see the thick arcs in Fig. 2(b)). On the other hand, for every $T \in \mathcal{T} \setminus \mathcal{T}'$, with $T = (a_i, b_j, c_p)$, we add to $\mathcal{P}$ the two paths (x_1, y_0, y_1) and (x_0, z_0, z_1) (see the thick arcs in Fig. 2(c)). Since $\mathcal{T}'$ is a solution of 3-Dimensional Matching, it holds that $|\mathcal{T}'| = n$, and thus $\mathcal{P} = 3n + 2(m - n) = n + 2m$, as required.

Conversely, suppose that G contains a subdivision of a $(k^* \times 4)$-spindle S. Since s and t are the only vertices in G with in-degree and out-degree at least k^*, respectively, necessarily they are the endpoints of S. Since $|V(G) \setminus \{s, t\}| = 3k^*$, it follows that $S \setminus \{s, t\}$ consists of a collection $\mathcal{P}$ of k^* vertex-disjoint 2-paths that covers all the vertices in $V(G) \setminus \{s, t\}$. Let H be the subdigraph in G associated with an arbitrary triple $T \in \mathcal{T}$, and consider $\mathcal{P} \cap H$. By construction of H, it follows that if $\mathcal{P} \cap H$ is not equal to one of the configurations corresponding to the thick arcs of Fig. 2(b) or Fig. 2(c), necessarily at least one vertex in $V(H)$ would not be covered by $\mathcal{P}$, a contradiction. Let $\mathcal{T}'$ be the set of triples in $\mathcal{T}$ such that the corresponding gadget H intersects $\mathcal{P}$ as in Fig. 2(b). It follows that $3|\mathcal{T}'| + 2(m - |\mathcal{T}'|) = |\mathcal{P}| = k^* = n + 2m$, and therefore $|\mathcal{T}'| = n$. Since all the 2-paths in $\mathcal{P}$ associated with the triples in $\mathcal{T}'$ are vertex-disjoint, we have that $\mathcal{T}'$ is a collection of n pairwise disjoint triples, hence a solution of 3-Dimensional Matching. $\square$

For a general $\ell > 4$, we define the digraph G in the same way, except that we subdivide the arcs outgoing from s exactly $\ell - 4$ times. The rest of the proof is essentially the same, and the result follows. $\square$

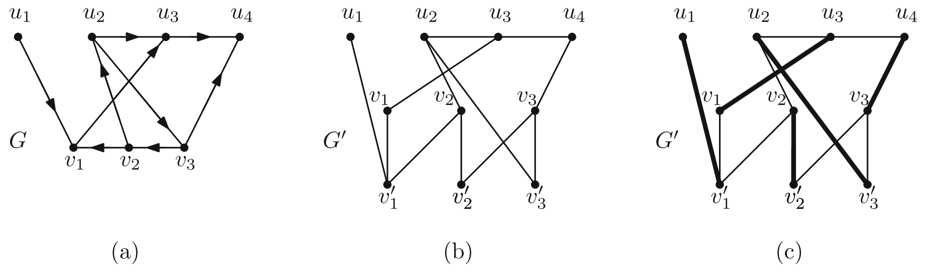


Fig. 3. (a) Digraph G with $X = \{u_1, u_2, u_3\}$ and $Y = \{u_3, u_4\}$. (b) Graph G' associated with G. (c) The thick edges define a matching of size five in G', corresponding to the two vertex-disjoint directed nontrivial paths (u_1, v_1, u_3) and (u_2, v_3, u_4) from X to Y in G.

We now turn to the cases that can be solved in polynomial time. We first need some ingredients to deal with the case $\ell = 3$, which is the most interesting one. Let G be a digraph and let X and Y be two subsets of $V(G)$. We say that a (simple) path P is *directed from X to Y* if P is a directed path with first vertex x and last vertex y such that $x \in X$ and $y \in Y$. The path P is *nontrivial* if its endpoints are distinct.

The following proposition will be the key ingredient in the proof of Theorem 7. Its proof is inspired by similar constructions given by Schrijver [19] on undirected graphs and by Kriesell [16] on directed graphs, usually called *vertex splitting procedure* (see [3, Sect. 5.9]). In fact, the conclusion of Proposition 1 can be also derived as a corollary of the main result in [16], noting that a polynomial-time algorithm can be extracted from that proof. We present a simpler proof here for completeness.

Proposition 1. *Let G be a digraph and let $X, Y \subseteq V(G)$. The maximum number of vertex-disjoint directed nontrivial paths from X to Y can be computed in polynomial time.*

Proof. Let $\mathcal{P}$ be any collection of vertex-disjoint directed nontrivial paths from X to Y in G. We can rebuild each path in $\mathcal{P}$ so that it has no internal vertices in $X \cup Y$. Therefore, we can assume G has no arcs to a vertex in $X \setminus Y$ or from a vertex in $Y \setminus X$.

Let G' be the undirected graph built from G as follows. The vertex set of G' is obtained from $V(G)$ by adding a copy v' of each vertex v not in $X \cup Y$. We build the edge set of G' starting from the empty set as follows. For every vertex v not in $X \cup Y$, add the edge $\{v, v'\}$. For each arc (u, v) in G, we add the edge $\{u, v\}$ if $v \in X \cup Y$ and the edge $\{u, v'\}$ otherwise. See Fig. 3(a)–(b) for an example.

Claim 2 ($\star$). *The digraph G contains a family of k vertex-disjoint directed nontrivial paths from X to Y if and only if G' has a matching of size $k + |V(G) \setminus (X \cup Y)|$.*

Claim 2 tells us that we can obtain a maximum number of vertex-disjoint nontrivial paths from X to Y in G by finding a maximum matching in the graph G', which can be done in polynomial time [9]. The proposition follows. □

The main algorithmic result of this section follows easily from Proposition 1.

Theorem 7 (⋆). *Let $\ell \leq 3$ be a fixed integer. The* Max $(\bullet \times \ell)$-Spindle Subdivision *problem can be solved in polynomial time.*

Using similar techniques, we can prove a generalization of Theorem 7.

Theorem 8 (⋆). *Given a digraph G and non-negative integers k_1, k_2, k_2, deciding whether G contains a subdivision of an $(\ell_1^1, \ldots, \ell_{k_1}^1, \ell_1^2, \ldots, \ell_{k_2}^2, \ell_1^3, \ldots, \ell_{k_3}^3)$-spindle such that, for $j \in [3]$ and $i \in [k_j]$, $\ell_i^j = j$, can be solved in polynomial time.*

3 Finding Subdivisions of 2-Spindles

In this section we focus on finding subdivisions of 2-spindles, and we prove Theorems 2 and 3. We focus here on the FPT algorithms, and the hardness results can be found in the full version. Our FPT algorithms for finding subdivisions of (ℓ_1, ℓ_2)-spindles are based on the technique of *representative families* introduced by Monien [18]. We use the improved version of this technique recently presented by Fomin et al. [10] and, more precisely, our algorithms and notation are inspired by the ones for Long Directed Cycle given in [10]. We start with some definitions introduced in [10] that can also be found in [8].

Two independent sets A, B of a matroid $\mathcal{M}$ *fit* if $A \cap B = \emptyset$ and $A \cup B$ is independent.

Definition 1. *Let $\mathcal{M}$ be a matroid and $\mathcal{A}$ be a family of sets of size p in $\mathcal{M}$. A subfamily $\mathcal{A}' \subseteq \mathcal{A}$ is said to q-*represent *$\mathcal{A}$ if for every set B of size q such that there is an $A \in \mathcal{A}$ that fits B, there is an $A' \in \mathcal{A}'$ that also fits B. If $\mathcal{A}'$ q-represents $\mathcal{A}$, we write $\mathcal{A}' \subseteq_{rep}^{q} \mathcal{A}$.*

3.1 Finding 2-Spindles with Large Total Size

We start with the algorithm to solve the problem of, given a digraph G and a positive integer ℓ, deciding whether there exist two strictly positive integers ℓ_1, ℓ_2 with $\ell_1 + \ell_2 = \ell$ such that G contains a subdivision of an (ℓ_1, ℓ_2)-spindle, running in time $2^{O(\ell)} \cdot n^{O(1)}$.

If a subdigraph S of G is a subdivision of an (ℓ_1, ℓ_2)-spindle, with $\min\{\ell_1, \ell_2\} \geq 1$ and $\ell_1 + \ell_2 = \ell$, we say that S is a *good spindle*. We may assume that $\max\{\ell_1, \ell_2\} \geq 2$, as otherwise the desired spindle is just an arc with multiplicity two, which can be detected in polynomial time by using a maximum flow algorithm.

The following simple observation, whose proof can be easily verified, will be crucially used by the algorithm that we propose in the sequel. See Fig. 4 for an illustration.

Lemma 1. *A digraph G has a good spindle if and only if there exist vertices u, u_1, u_2, v, integers ℓ_1, ℓ_2 with $\min\{\ell_1, \ell_2\} \geq 1$ and $\ell_1 + \ell_2 = \ell$, a (u, u_1)-path P_1^u on ℓ_1 vertices, a (u, u_2)-path P_2^u on ℓ_2 vertices, a (u_1, v)-path P_1^v, and a (u_2, v)-path P_2^v such that $V(P_1^u) \cap V(P_2^u) = \{u\}$, $V(P_1^v) \cap V(P_2^v) = \{v\}$, $V(P_1^u) \cap V(P_1^v) = \{u_1\}$, $V(P_2^u) \cap V(P_2^v) = \{u_2\}$, and, if $\min\{\ell_1, \ell_2\} \geq 2$, $V(P_1^u) \cap V(P_2^v) = V(P_2^u) \cap V(P_1^v) = \emptyset$.*

In the above lemma, note that if $\min\{\ell_1, \ell_2\} = 1$ then one of the paths P_1^u and P_2^u, say P_1^u, may be degenerate to vertex u, and in that case we have that $u_1 = u$.

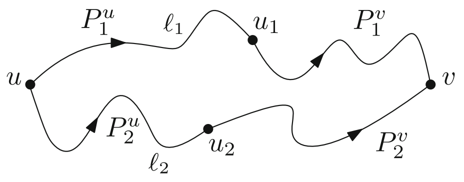


Fig. 4. Illustration of the vertices and paths described in Lemma 1.

Motivated by Lemma 3, for every triple of vertices $u, u_1, u_2 \in V(G)$ and positive integers ℓ_1, ℓ_2, we define

$$\mathcal{S}_{u,u_1,u_2}^{\ell_1,\ell_2} = \Big\{X : S \subseteq V(G), |X| = \ell_1 + \ell_2 - 1, \text{ and } G[X] \text{ contains a } (u, u_1)\text{-path } P_1^u \text{ on } \ell_1 \text{ vertices and a } (u, u_2)\text{-path } P_2^u \text{ on } \ell_2 \text{ vertices such that } V(P_1^u) \cap V(P_2^u) = \{u\}\Big\}.$$

The key idea is to compute efficiently a small family of subsets of $V(G)$ that *represents* the above sets, which are too large for our purposes. More precisely, for every triple of vertices $u, u_1, u_2 \in V(G)$ and positive integers ℓ_1, ℓ_2, q with $\ell_1, \ell_2 \leq \ell$ and $q \leq 2\ell - (\ell_1 + \ell_2)$, we will compute in time $2^{O(\ell)} \cdot n^{O(1)}$ a q-representative family $\widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q} \subseteq_{\text{rep}}^q \mathcal{S}_{u,u_1,u_2}^{\ell_1,\ell_2}$.

As in [10], the matroid with respect to which we will define the above q-representative family $\widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q}$ is the uniform matroid with ground set $V(G)$ and rank $\ell + q$.

We defer the computation of the above q-representative families in time $2^{O(\ell)} \cdot n^{O(1)}$ to the full version, and assume now that we already have these families at hand. The following lemma states that they are enough to find the desired good spindle.

Lemma 2. *If G contains a good spindle, then there exist vertices u, u_1, u_2, v, integers ℓ_1, ℓ_2 with $\min\{\ell_1, \ell_2\} \geq 1$ and $\ell_1 + \ell_2 = \ell$, a set $\widehat{S}_u \in \widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q}$ with $q \leq \ell - 1$, a (u_1, v)-path P_1^v, and a (u_2, v)-path P_2^v such that $V(P_1^v) \cap V(P_2^v) = \{v\}$ and $\widehat{S}_u \cap (V(P_1^v) \cup V(P_2^v)) = \{u_1, u_2\}$.*

Proof. Let S be a good spindle in G with minimum number of vertices, which exists by hypothesis, and let u and v be the tail and the head of S, respectively. Let $P_1^u = (u, \dots, u_1)$ and $P_2^u = (u, \dots, u_2)$ be two subdipaths in S outgoing from u, on ℓ_1 and ℓ_2 vertices, respectively, with $\ell_1 + \ell_2 = \ell$. Let also $P_1^v = (u_1, \dots, v)$ and $P_2^v = (u_2, \dots, v)$ be the two subdipaths in S from u_1 and u_2 to v, respectively (see Fig. 4). Let $S_u = V(P_1^u) \cup V(P_2^u)$, and note that $S_u \in \mathcal{S}_{u,u_1,u_2}^{\ell_1,\ell_2}$.

In order to apply the properties of q-representative families, we define a vertex set $B \subseteq V(S)$ as follows. If $|V(S) \setminus S_u| \leq \ell - 2$, let $B = V(S) \setminus S_u$. Otherwise, let B be the union of two subdipaths $P_1^B = (v_1, \dots, v)$ and $P_2^B = (v_2, \dots, v)$ in S with $V(P_1^B) \cap V(P_2^B) = \{v\}$ and $|V(P_1^B) \cup V(P_2^B)| = \ell - 1$. Note that there may be several choices for the lengths of P_1^B and P_2^B, as far as their joint number of vertices is equal to $\ell - 1$. Note also that P_1^B (resp. P_2^B) is a subdipath of P_1^v (resp. P_2^v).

Let $q = |B| \leq \ell - 1$. Since $S_u \in \mathcal{S}_{u,u_1,u_2}^{\ell_1,\ell_2}$ and $S_u \cap B = \emptyset$, by definition of q-representative family there exists $\widehat{S}_u \in \widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q}$ such that $\widehat{S}_u \cap B = \emptyset$. We claim that $\widehat{S}_u \cap (V(P_1^v) \cup V(P_2^v)) = \{u_1, u_2\}$, which concludes the proof of the lemma. If $|B| \leq \ell - 2$, the claim follows easily as $\widehat{S}_u \cap B = \emptyset$ and B contains all the vertices in $V(S) \setminus S_u$. Suppose henceforth that $|B| \geq \ell - 1$, and let $\widehat{P}_1^u$ and $\widehat{P}_2^u$ be the two paths in $G[\widehat{S}_u]$ with $V(\widehat{P}_1^u) \cap V(\widehat{P}_1^u) = \{u\}$. Assume for contradiction that $(\widehat{S}_u \cap (V(P_1^v) \cup V(P_2^v))) \setminus \{u_1, u_2\} \neq \emptyset$, and we distinguish two cases.

Suppose first that each of the paths $\widehat{P}_1^u$ and $\widehat{P}_2^u$ intersects exactly one of the paths P_1^v and P_2^v. By hypothesis, there exists a vertex $w \in (\widehat{S}_u \cap (V(P_1^v) \cup V(P_2^v))) \setminus \{u_1, u_2\}$, and suppose without loss of generality that $w \in V(\widehat{P}_1^u) \cap V(P_1^v)$; see Fig. 5(a) for an illustration. We define a good spindle $\widehat{S}$ in G as follows. The tail and head of $\widehat{S}$ are vertices u and v, respectively. The first path of $\widehat{S}$ starts at u, follows $\widehat{P}_1^u$ until its first intersection with P_1^v (vertex w in Fig. 5(a)), which is distinct from u_1 by hypothesis, and then follows P_1^v until v. The second path of $\widehat{S}$ starts at u, follows $\widehat{P}_2^u$ until its first intersection with P_2^v, which may be vertex u_2, and then follows P_2^v until v. Since $|B| \geq \ell - 1$ and each of $\widehat{P}_1^u$ and $\widehat{P}_2^u$ intersects exactly one of P_1^v and P_2^v, it follows that $\widehat{S}$ is indeed a good spindle. On the other hand, since $|V(\widehat{P}_1^u) \cup V(\widehat{P}_2^u)| = |V((P_1^u) \cup V(P_2^u)|$ and vertex w comes strictly after u_1 in P_1^v, it follows that the first path of $\widehat{S}$ is strictly shorter than the corresponding path of S, while the second one is not longer. Therefore, $|V(\widehat{S})| < |V(S)|$, a contradiction to the choice of S.

Suppose now that one of the paths $\widehat{P}_1^u$ and $\widehat{P}_2^u$, say $\widehat{P}_1^u$, intersects both P_1^v and P_2^v. Without loss of generality, suppose that, starting from u, $\widehat{P}_1^u$ meets P_1^v before than P_2^v. Let w_1 and w_2 be vertices of $\widehat{P}_1^u$ such that $w_1 \in V(P_1^v)$, $w_2 \in V(P_2^v)$, and there is no vertex of $\widehat{P}_1^u$ between w_1 and w_2 that belongs to $V(P_1^v) \cup V(P_2^v)$; see Fig. 5(b) for an illustration. We define a good spindle $\widehat{S}$ in G as follows. The tail and head of $\widehat{S}$ are vertices w_1 and v, respectively. The first path of $\widehat{S}$ starts at w_1 and follows P_1^v until v. The second path of $\widehat{S}$ starts at w_1, follows $\widehat{P}_1^u$ until w_2, and then follows P_2^v until v. By the choice of w_1 and w_2

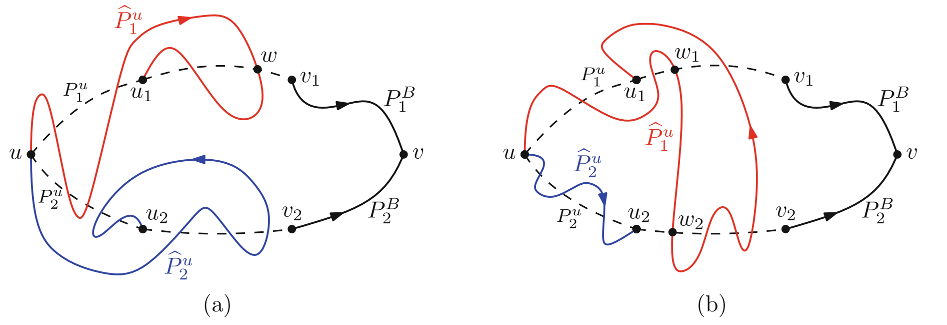


Fig. 5. Illustration of the two cases in the proof of Lemma 2.

and since $|B| \geq \ell - 1$, it follows that $\widehat{S}$ is indeed a good spindle. On the other hand, by construction $|V(\widehat{S})| \leq |V(S)| - |V(\widehat{P}_2^u)| < |V(S)|$, contradicting again the choice of S. $\square$

Wrapping up the algorithm. We finally have all the ingredients to describe our algorithm, which proceeds as follows. First, for every triple of vertices $u, u_1, u_2 \in V(G)$ and positive integers ℓ_1, ℓ_2, q with $\ell_1, \ell_2 \leq \ell$ and $q \leq 2\ell - (\ell_1 + \ell_2)$, we compute, as explained in the full version, a q-representative family $\widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q} \subseteq_{\text{rep}}^{q} \mathcal{S}_{u,u_1,u_2}^{\ell_1,\ell_2}$ of size $2^{O(\ell)}$ in time $2^{O(\ell)} \cdot n^{O(1)}$. Then the algorithm checks, for each $u, u_1, u_2, v \in V(G)$, integers ℓ_1, ℓ_2, q with $\min\{\ell_1, \ell_2\} \geq 1$, $\ell_1 + \ell_2 = \ell$, and $q \leq \ell - 1$, and set $S \in \widehat{\mathcal{S}}_{u,u_1,u_2}^{\ell_1,\ell_2,q}$, whether G contains a (u_1, v)-path P_1^v and a (u_2, v)-path P_2^v such that $V(P_1^v) \cap V(P_2^v) = \{v\}$ and $S \cap (V(P_1^v) \cup V(P_2^v)) = \{u_1, u_2\}$. Note that the latter check can be easily performed in polynomial time by a flow algorithm [3]. The correctness of the algorithm follows directly from Lemmas 1 and 2, and its running time is $2^{O(\ell)} \cdot n^{O(1)}$, as claimed. In order to keep the exposition as simple as possible, we did not focus on optimizing either the constants involved in the algorithm or the degree of the polynomial factor. Explicit small constants can be derived by following the details in [10], or alternatively in [21].

3.2 Finding 2-Spindles with Two Specified Lengths

We now turn to the problem of finding 2-spindles with two specified lengths. Namely, given a digraph G and two integers ℓ_1, ℓ_2 with $\ell_2 \geq \ell_1 \geq 1$, our objective is to decide whether G contains a subdivision of an (ℓ_1, ℓ_2)-spindle in time $2^{O(\ell_2)} \cdot n^{O(\ell_1)}$. Note that this problem differs from the one considered in Sect. 3.1, as now we specify *both* lengths of the desired spindle, instead of just its total size. Our approach is similar to the one presented in Sect. 3.1, although some more technical ingredients are needed, and we need to look at the problem from a slightly different point of view.

In this section, we say that a subdigraph S of G is a *good spindle* if it is a subdivision of an (ℓ_1, ℓ_2)-spindle. We may again assume that $\max\{\ell_1, \ell_2\} \geq 2$.

The following lemma plays a similar role as Lemma 1, but now we will exploit the fact that our algorithm can afford to guess the first ℓ_1 vertices in the "short" path. Its proof is also easy to verify. See Fig. 6 for an illustration.

Lemma 3. *A digraph G has a good spindle if and only if there exist vertices u, u', v, a (u, v)-path P_1 of length at least ℓ_1, a (u, u')-path P_2^u on ℓ_2 vertices, and a (u', v)-path P_2^v such that $V(P_1) \cap V(P_2^u) = \{u\}$, $V(P_1) \cap V(P_2^v) = \{v\}$, and $V(P_2^u) \cap V(P_2^v) = \{u'\}$.*

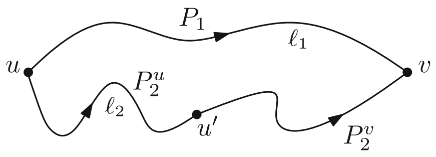


Fig. 6. Illustration of the vertices and paths described in Lemma 3.

The main difference with respect to Sect. 3.1 is that now we will only represent the candidates for the first ℓ_2 vertices of the "long" path, denoted by $V(P_2^u)$ in Lemma 3. To this end, we define, similarly to [10], the following set for every pair of vertices $u, u' \in V(G)$ and positive integer ℓ_2:

$$\mathcal{P}_{u,u'}^{\ell_2} = \Big\{X : \; S \subseteq V(G), |X| = \ell_2, \text{ and } G[X] \text{ contains a } (u, u')\text{-path on } \ell_2 \text{ vertices}\Big\}.$$

The above sets are exactly the same as those defined by Fomin et al. [10] to solve the Long Directed Cycle problem. Therefore, we can just apply [10, Lemma 5.2] and compute, for every pair of vertices $u, u' \in V(G)$ and positive integers ℓ_2, q with $q \le \ell_1 + \ell_2 \le 2\ell_2$, a q-representative family $\widehat{\mathcal{P}}_{u,u'}^{\ell_2,q} \subseteq_{\text{rep}}^{q} \mathcal{P}_{u,u'}^{\ell_2}$ of size $2^{O(\ell_2)}$ in time $2^{O(\ell_2)} \cdot n^{O(1)}$.

Now we would like to state the equivalent of Lemma 2 adapted to the new representative families. However, it turns out that the families $\widehat{\mathcal{P}}_{u,u'}^{\ell_2,q}$ do not yet suffice in order to find the desired spindle. To circumvent this cul-de-sac, we use the following trick: we first try to find "short" spindles using the color-coding technique of Alon et al. [1], and if we do not succeed, we can guarantee that all good spindles have at least one "long" path. In this situation, we can prove that the families $\widehat{\mathcal{P}}_{u,u'}^{\ell_2,q}$ are indeed enough to find a good spindle. More precisely, a good spindle S is said to be *short* if both its paths have at most $2\ell_2$ vertices, and it is said to be *long* otherwise. Note that the following lemma only applies to digraphs without good short spindles.

Lemma 4 ($\star$). *Let G be a digraph containing no good short spindles. If G contains a good long spindle, then there exist vertices u, u', v, a (u, v)-path P_1 of length at least ℓ_1, a (u, u')-path $\widehat{P}_2^u$ on ℓ_2 vertices such that $V(\widehat{P}_2^u) \in \widehat{\mathcal{P}}_{u,u'}^{\ell_2,q}$ with $q = \ell_1 + \ell_2 - 1$, and a (u', v)-path P_2^v such that $V(P_1) \cap V(\widehat{P}_2^u) = \{u\}$, $V(P_1) \cap V(P_2^v) = \{v\}$, and $V(\widehat{P}_2^u) \cap V(P_2^v) = \{u'\}$.*

Wrapping up the algorithm. We start by trying to find good small spindles. Namely, for every pair of integers ℓ_1', ℓ_2' with $\ell_1 \leq \ell_1' \leq 2\ell_2$ and $\ell_2 \leq \ell_2' \leq 2\ell_2$, we test whether G contains an (ℓ_1', ℓ_2')-spindle as a *subgraph*, by using the color-coding technique of Alon et al. [1]. Since the treewidth of an undirected spindle is two, this procedure takes time $2^{O(\ell_2)} \cdot n^{O(1)}$.

If we succeed, the algorithm stops. Otherwise, we can guarantee that G does not contain any good short spindle, and therefore we are in position to apply Lemma 4. Before this, we first compute, for every pair of vertices $u, u' \in V(G)$ and positive integers ℓ_2, q with $q \leq \ell_1 + \ell_2 \leq 2\ell_2$, a q-representative family $\widehat{\mathcal{P}}^{\ell_2,q}_{u,u'} \subseteq^q_{\text{rep}} \mathcal{P}^{\ell_2}_{u,u'}$ of size $2^{O(\ell_2)}$ in time $2^{O(\ell_2)} \cdot n^{O(1)}$, using [10, Lemma 5.2].

Now, for each path $\widehat{P}^u_2$ such that $V(\widehat{P}^u_2) \in \widehat{\mathcal{P}}^{\ell_2,q}_{u,u'}$, with $q = \ell_1 + \ell_2 - 1$, we proceed as follows. By Lemmas 3 and 4, it is enough to guess a vertex $v \in V(G)$ and check whether G contains a (u, v)-path P_1 of length at least ℓ_1, and a (u', v)-path P^v_2 such that $V(P_1) \cap V(\widehat{P}^u_2) = \{u\}$, $V(P_1) \cap V(P^v_2) = \{v\}$, and $V(\widehat{P}^u_2) \cap V(P^v_2) = \{u'\}$. In order to do so, we apply brute force and we guess the first ℓ_1 vertices of P_1 in time $n^{O(\ell_1)}$. Let these vertices be $u, u_2, \ldots, u_{\ell_1}$. All that remains is to test whether the graph $G \setminus \{u_2, \ldots, u_{\ell_1 - 1}\} \setminus (V(\widehat{P}^u_2) \setminus \{u'\})$ contains two internally vertex-disjoint paths from u_{ℓ_1} and u' to u, which can be done in polynomial time by using a flow algorithm [3]. The correctness of the algorithm follows by the above discussion, and its running time is $2^{O(\ell_2)} \cdot n^{O(\ell_1)}$, as claimed. Again, we did not focus on optimizing the constants involved in the algorithm.

4 Conclusions

We studied the complexity of several problems consisting in finding subdivisions of spindles on digraphs. For a general spindle F, we do not know if finding a subdivision of F is FPT on general digraphs parameterized by $|V(F)|$, although we believe that it is indeed the case. As a partial result, one could try to prove that, for a *fixed* value of $\ell \geq 4$, finding a subdivision of a $(k \times \ell)$-spindle is FPT parameterized by k (the problem is NP-hard by Theorem 1).

The above question is open even if the input digraph is acyclic (note that Theorem 4 does *not* answer this question), or even if F is a 2-spindle. Concerning 2-spindles, one may try use the technique we used to prove Theorems 2 and 3, based on representative families in matroids. However, the technique does not seem to be easily applicable when the parameter is the total size of a prescribed 2-spindle. Namely, using the terminology from Sect. 3.2, the bottleneck is to find spindles that have one "short" and one "long" path. On the other hand, generalizing this technique to spindles with more than two paths seems pretty complicated.

Finally, it may be possible that the trick used by Zehavi [22] to avoid the use of representative families to solve Long Directed Cycle can be adapted to our setting as well.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM **42**(4), 844–856 (1995)
2. Araújo, J., Campos, V.A., Maia, A.K., Sau, I., Silva, A.: On the complexity of finding internally vertex-disjoint long directed paths. CoRR, abs/1706.09066 (2017)
3. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer, London (2008). https://doi.org/10.1007/978-1-84800-998-1
4. Bang-Jensen, J., Havet, F., Maia, A.K.: Finding a subdivision of a digraph. Theoret. Comput. Sci. **562**, 283–303 (2015)
5. Benhocine, A., Wojda, A.P.: On the existence of specified cycles in a tournament. J. Graph Theory **7**(4), 469–473 (1983)
6. Brewster, R.C., Hell, P., Pantel, S.H., Rizzi, R., Yeo, A.: Packing paths in digraphs. J. Graph Theory **44**(2), 81–94 (2003)
7. Cohen, N., Havet, F., Lochet, W., Nisse, N.: Subdivisions of oriented cycles in digraphs with large chromatic number. CoRR, abs/1605.07762 (2016)
8. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
9. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173, 4th edn. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-662-53622-3
10. Fomin, F.V., Lokshtanov, D., Panolan, F., Saurabh, S.: Efficient computation of representative families with applications in parameterized and exact algorithms. J. ACM **63**(4), 29:1–29:60 (2016)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
12. Grohe, M., Kawarabayashi, K., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. In: Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC), pp. 479–488 (2011)
13. Havet, F., Maia, A.K., Mohar, B.: Finding a subdivision of a prescribed digraph of order 4. J. Graph Theory (to appear). https://doi.org/10.1002/jgt.22174
14. Itai, A., Perl, Y., Shiloach, Y.: The complexity of finding maximum disjoint paths with length constraints. Networks **12**(3), 277–286 (1982)
15. Kim, R., Kim, S.-J., Ma, J., Park, B.: Cycles with two blocks in k-chromatic digraphs. CoRR, abs/1610.05839 (2016)
16. Kriesell, M.: Disjoint A-paths in digraphs. J. Comb. Theory Ser. B **95**(1), 168–172 (2005)
17. Metzlar, A.: Disjoint paths in acyclic digraphs. J. Comb. Theory Ser. B **57**(2), 228–238 (1993)
18. Monien, B.: How to find long paths efficiently. Ann. Discret. Math. **25**, 239–254 (1985)
19. Schrijver, A.: A short proof of Mader's $\mathcal{S}$-paths theorem. J. Comb. Theory Ser. B **82**(2), 319–321 (2001)
20. Sedgewick, R., Wayne, K.: Algorithms, 4th edn. Addison-Wesley, Boston (2011)
21. Shachnai, H., Zehavi, M.: Representative families: a unified tradeoff-based approach. J. Comput. Syst. Sci. **82**(3), 488–502 (2016)
22. Zehavi, M.: A randomized algorithm for long directed cycle. Inf. Process. Lett. **116**(6), 419–422 (2016)

Algorithms and Hardness Results for Nearest Neighbor Problems in Bicolored Point Sets

Sandip Banerjee[1], Sujoy Bhore[2](✉), and Rajesh Chitnis[3]

[1] Indian Statistical Institute, Kolkata, India
sandip.ndp@gmail.com
[2] Ben-Gurion University of the Negev, Beersheba, Israel
sujoy.bhore@gmail.com
[3] Department of Computer Science, University of Warwick, Coventry, UK
rajeshchitnis@gmail.com

Abstract. In the context of computational supervised learning, the primary objective is the classification of data. Especially, the goal is to provide the system with "training" data and design a method which uses the training data to classify new objects with the correct label. A standard scenario is that the examples are points from a metric space, and "nearby" points should have "similar" labels. In practice, it is desirable to reduce the size of the training set without compromising too much on the ability to correctly label new objects. Such subsets of the training data are called as edited sets. Wilfong [SOCG '91] defined two types of edited subsets: consistent subsets (those which correctly label all objects from the training data) and selective subsets (those which correctly label all new objects the same way as the original training data). This leads to the following two optimization problems:

- $\underline{k\text{-MCS-}(\mathcal{X})}$: Given k sets of points $P_1, P_2, \ldots, P_k$ in a metric space $\mathcal{X}$, the goal is to choose subsets of points $P'_i \subseteq P_i$ for $i = 1, 2, \ldots, k$ such that $\forall\ p \in P_i$ its nearest neighbor among $\bigcup_{j=1}^{k} P'_j$ lies in P'_i for each $i \in [k]$ while minimizing (Note that we also enforce the condition $|P'_i| \geq 1\ \forall\ i \in [k]$.) the quantity $\sum_{i=1}^{k} |P'_i|$.
- $\underline{k\text{-MSS-}(\mathcal{X})}$: Given k sets of points $P_1, P_2, \ldots, P_k$ in a metric space $\mathcal{X}$, the goal is to choose subsets of points $P'_i \subseteq P_i$ for $i = 1, 2, \ldots, k$ such that $\forall\ p \in P_i$ its nearest neighbor among $\left(\bigcup_{j=1, j\neq i}^{k} P_j\right) \cup P'_i$ lies in P'_i for each $i \in [k]$ while minimizing (Note that we again enforce the condition $|P'_i| \geq 1\ \forall\ i \in [k]$.) the quantity $\sum_{i=1}^{k} |P'_i|$.

While there have been several heuristics proposed for these two problems in the computer vision and machine learning community, the only theoretical results for these problems (to the best of our knowledge) are due to Wilfong [SOCG '91] who showed that both 3-MCS-($\mathbb{R}^2$) and

S. Bhore—Supported by the Lynn and William Frankel Center for Computer Science.
R. Chitnis—Supported by ERC grant CoG 647557 "Small Summaries for Big Data". Part of this work was done when the author was at the Weizmann Institute of Science and supported by Israel Science Foundation grant #897/13.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 80–93, 2018.
https://doi.org/10.1007/978-3-319-77404-6_7

2-MSS-($\mathbb{R}^2$) are NP-complete. We initiate the study of these two problems from a theoretical perspective, and obtain several algorithmic and hardness results.

On the algorithmic side, we first design an $O(n^2)$ time exact algorithm and $O(n \log n)$ time 2-approximation for the 2-MCS-($\mathbb{R}$) problem, i.e., the points are located on the real line. Moreover, we show that the exact algorithm also extends to the case when the points are located on the circumference of a circle. Next, we design an $O(r^2)$ time online algorithm for the 2-MCS-($\mathbb{R}$) problem such that $r < n$, where n is the set of points and r is an integer. Finally, we give a PTAS for the k-MSS-($\mathbb{R}^2$) problem. On the hardness side, we show that both the 2-MCS and 2-MSS problems are NP-complete on graphs. Additionally, the problems are W[2]-hard parameterized by the size k of the solution. For points on the Euclidean plane, we show that the 2-MSS problem is contained in W[1]. Finally, we show a lower bound of $\Omega(\sqrt{n})$ bits for the storage of any (randomized) algorithm which solves both 2-MCS-($\mathbb{R}$) and 2-MSS-($\mathbb{R}$).

1 Introduction

Pattern recognition is the science of making inferences from perceptual data, using tools from statistics, probability, computational geometry, machine learning, signal processing and algorithm design. Pattern recognition has many applications in speech, handwriting, object and character recognition in vision research [4,6]. Classifying a new object according to the nearest neighbor rule is one of the important problems in pattern recognition. There are two classification methods in pattern recognition: supervised and unsupervised classification. The idea behind supervised learning is to present the system with sufficient training data, i.e., examples of objects with the correct labels. More specifically, a training set is a set of colored points called examples and the goal is to correctly color the query point using the training set. The principal aim then is to design a classifying method for the system to use this training to determine the label of new objects presented to it. This has many applications in speech, handwriting, object and character recognition in vision research [4,6]. There are two main issues in formalizing a classifying method which decides the labels for new examples are *choice of representation* of the examples, and more importantly the rule for deciding the likely label of a given example. A standard choice for representation of an example is by a point in k-dimensional Euclidean space (or more generally, any metric space). Another reasonable assumption is that objects which are "near" to each other should have same labels. This led to the Nearest Neighbor Decision Rule (NN): simply assign each new object the same label as the object in the training data that is nearest to it. The issue with the NN rule is that one needs to have a large number of samples in the training data to keep the error probability low. Not only does this lead to storage issues, but also the time required to find the nearest neighbor in the training data also turns out to be expensive. Hence, it is a natural goal to try to reduce the size of the training set, but at the same time try to not compromise too much on the ability to correctly label new objects. These subsets of the training sets are called as *edited* sets by Wilfong [3]. A first step in this direction is to compute a small edited set which correctly labels all the examples from the original training set! Formally, this leads to the following problem:

k-MCS($\mathcal{X}$)
Input: k sets of points $P_1, P_2, \ldots, P_k$ in a metric space $\mathcal{X}$
Output: Choose subsets of points $P'_i \subseteq P_i$ for $i = 1, 2, \ldots, k$ such that

- For any point $p \in P_i$ its nearest neighbor among $\bigcup_{j=1}^{k} P'_j$ lies in P'_i for each $i \in [k]$.
- $\sum_{i=1}^{k} |P'_i|$ is minimized.
- $|P'_i| > 0$ for each $i \in [k]$.

The set $\bigcup_{i=1}^{k} P'_i$ is called as a minimum consistent subset (`mcs`). One way to view this problem is to imagine $P_i \setminus P'_i$ as the clients, and the points in P'_i as the facility stations of type i, for all $i = 1, 2, \ldots, k$. A client always gets service from its nearest facility station. The objective is to choose facility stations such that a client of type i gets service from a facility station of type i. When the domain $\mathcal{X}$ is a graph (and distances are shortest paths in the graph) then we call the problem simply as k-MCS. The first heuristic for computing a `mcs` was designed by Hart [5] in 1968 and was called as Condensed Nearest Neighbor Rule (CNN). Although CNN had a worst case running time of $O(n^3)$ where n is the total number of objects, there was no guarantee on the optimality of the size of the consistent subset found. After this, there were several attempts at developing heuristics, but they either did not guarantee optimality [7] or had exponential running time [12]. Finally Wilfong [3] showed that the 3-MCS($\mathbb{R}^2$) is NP-complete. The proof is based on the NP completeness proof of the DISC-COVER problem [8]. A stronger[1] requirement is when we want a small edited subset which labels all new objects the same way as the original training set. Formally, this leads to the following problem:

k-MSS($\mathcal{X}$)
Input: k sets of points $P_1, P_2, \ldots, P_k$ in $\mathcal{X}$
Output: Choose subsets of points $P'_i \subseteq P_i$ for $i = 1, 2, \ldots, k$ such that

- For any point $p \in P_i$ its nearest neighbor among $(\bigcup_{j=1, j\neq i}^{k} P_j) \cup P'_i$ lies in P'_i for each $i \in [k]$.
- $\sum_{i=1}^{k} |P'_i|$ is minimized.
- $|P'_i| > 0$ for each $i \in [k]$.

Wilfong [3] showed that 2-MSS($\mathbb{R}^2$) is NP-complete. To the best of our knowledge, this is the only known theoretical result about this problem. The 2-colored (bichromatic) input has appeared in many geometric problems [13, 14].

[1] Note that for MCS we only want to be correct on the training set.

Results and Organization: In this paper, we reinitiate the study of the k-MCS-($\mathcal{X}$)and k-MSS-($\mathcal{X}$) problems from a theoretical perspective. Our goal is to study the tractability landscape of these problems with respect to various paradigms such as polynomial time algorithms, NP-hardness, parameterized complexity and streaming algorithms. We were able to find tractable special cases of these problems, and also determine in some scenarios in which these problems provably intractable. Mostly, we focus on the case when the points only have two labels/colors, i.e., $k = 2$. We now state our results below formally.

On the algorithmic side, in Sect. 2.1, we design an $O(n^2)$ exact algorithm for the 2-MCS-($\mathbb{R}$), i.e., the points are located on the real line. Next we show that this algorithm also extends to the case when the points are located on the circumference of a circle. Further, we construct a $O(r^2)$ time online algorithm for the MCS problem for points on a real line such that $r < n$, where n is the set of points and r is an integer. Finally, in Sect. 2.2, we give a PTAS for the MSS problem by adopting the hitting set algorithm from [10].

On the hardness side, we prove in Sect. 3.1 that both the MCS and MSS problems are NP-complete on graphs. Additionally, the problems are W[2]-hard parameterized by the size k of the solution, and under the ETH the brute force $n^{O(k)}$ algorithm is optimal since there is no $f(k) \cdot n^{o(k)}$ algorithm for any function f. Moreover, we show that for points on the Euclidean plane 2-MSS problem (namely 2-MSS-($\mathbb{R}^2$)) is contained in W[1]. Finally, we show a lower bound of $\Omega(\sqrt{n})$ bits for the storage of any (randomized) algorithm which solves both 2-MCS-($\mathbb{R}$) and 2-MSS-($\mathbb{R}$).

2 Algorithmic Results

2.1 Polynomial Time Algorithms for MCS Problem on the Real Line

Let $P = R \cup B$ be a set of points on a real line $\mathcal{L}$, where the points in the set R (resp. B) are colored *red* (resp. *blue*). A point $\alpha \in P$ is called *dominating* on a point $\beta \in P$ if α is the nearest neighbor of β. The objective is to compute a *consistent* subset $\{R' \cup B'\} \subseteq \{R \cup B\}$, where $R' \subseteq R$ and $B' \subseteq B$, such that each point in R (resp. B) is dominated by a point from R' (resp. B'). Here we address the *minimum consistent subset* (shortly `mcs`) problem where the goal is to minimize the cardinality of such consistent set.

We denote a consecutive sequence of monochromatic points as *blocks*. Hence a set of bichromatic points on a real line induces to a sequence of blocks of alternating colors. Let $D_1, \ldots, D_m$ are the blocks listed by visiting the points of $P = R \cup B$ from left to right. Let $S = R' \cup B'$ be a feasible `mcs`. A block D_i is called *self-dominating* with respect to a point set $P_i \subseteq S$ if the point set P_i belongs to th block D_i and for any point $q_i \in D_i \setminus P_i$ is dominated by one of the points in the set P_i.

Lemma 1. *Given a set $P = R \cup B$ of bi-colored points on a real line, in any feasible* `mcs` *of S, every monochromatic block is self-dominating.*

Proof. For the sake of contradiction, suppose there exists a monochromatic block D_i that does not contain any point p which is part of the feasible solution that dominates every other point of D_i. This implies that the points in D_i are dominated by a point in $D_{i-1} \cup D_{i+1}$ whereas the color of the points in $D_{i-1} \cup D_{i+1}$ are different from the color of the points in D_i implying S is not a feasible MCS of $P = R \cup B$. Thereby we conclude the proof. □

Lemma 2. *In any feasible* `mcs`*, for each block D_i at most 2 dominators suffice and specifically for the left-most and right-most block only 1 dominator suffices.*

Proof. The first part follows from the fact that if we choose only the left-most and right-most element in each block, then all other elements in that block will be dominated by those two elements.

The second part is also obvious since only the rightmost element in the first block suffices irrespective of the position of the left-dominator in the second block. Similar argument is also applicable for the last block. □

From Lemmas 1 and 2, we have the following theorem.

Theorem 1. *There exists $O(n \log n)$ time, 2-approximation algorithm for the* `mcs` *problem for a set of bi-colored points on a real line.*

Proof. Sort the input point set P to form the blocks. By Lemma 1, the lower bound on the size of the MCS is χ, where χ is the number of blocks created with the points in P. Our algorithm chooses the elements in S as follows: For a block D_i, (i) if $|D_i| = 1$, then include the only point of D_i in S, and (ii) if $|D_i| > 1$, then include the left-most and the right-most points of D_i on the line $\mathcal{L}$ in S. Thus, we have $|S| \leq 2\chi$. □

Exact $O(n^2)$ time algorithm using dynamic programming: Let us assume that the points in P are given in an array $\mathcal{A} = \{p_1, p_2, \ldots, p_n\}$ in sorted order. Let $D_1, \ldots, D_m$ are the blocks listed by visiting the points from left to right. Consider any two consecutive blocks D_i and D_{i+1} and a pair of points $a \in D_i$ and $b \in D_{i+1}$ of different colors. We denote (a, b) as a *valid pair* if no point in D_{i+1} is dominated by a, and no point in D_i is dominated by b. Formally, $V_i = \{(a, b) \mid a \in D_i, b \in D_{i+1}, (a, b) \text{ is a valid pair}\}$. Recall that, every block has at most two dominators, we shall refer them as the *left* and the *right* dominators. If a block has a single dominator, we shall refer it as *right* dominator of the block. We solve the problem using a dynamic programming (DP) technique. The idea is we fix a dominator in the rightmost block D_m (Ref. Lemma 2), and solve the subproblem involving blocks $D_1, \ldots, D_{m-1}$.

Let $\mathcal{A}[a, k]$ be the optimum solution for the consecutive blocks $D_1, \ldots, D_a$ provided $p_k \in D_a$ is the right dominator of D_a. Given the blocks $D_1, \ldots, D_m$, the optimum solution is $OPT = \min_{k|(k,l)\in V_{m-1}} \mathcal{A}[m-1,k]+1$, since D_m has only one dominator (Ref. Lemma 2).

While computing $\mathcal{A}[m-1,k]$ (where $p_k \in D_{m-1}$), we distinguish between the following cases based on the fact that D_{m-1} either has 1 or 2 dominators.

- If D_{m-1} has only 1 dominator, then $\mathcal{A}[m-1,k]=U$, where $U = \min_{i\in D_{m-1}} \{(\mathcal{A}[m-2,i]+1)|(i,k)\in V_{m-2}\}$.
- If D_{m-1} has 2 dominators then $\mathcal{A}[m-1,k]=V$, where $V = \min_{i\in D_{m-2},\ j\in D_{m-1}} \{(\mathcal{A}[m-2,i]+2)|j<k \ \&\ (i,j)\in V_{m-2}\}$.
- Hence $\mathcal{A}[m-1,k]=\min(U,V)$.

In general, we can write $\mathcal{A}[\alpha,k]=1$ if $\alpha=1$; else, $\mathcal{A}[\alpha,k]=\min(U,V)$, where

- $U = \min_{i\in D_{\alpha-1}} \{(\mathcal{A}[\alpha-1,i]+1)|(i,k)\in V_{\alpha-1}\}$, if D_α has only one dominator
- $V = \min_{i\in D_{\alpha-1},j\in D_\alpha} \{(\mathcal{A}[\alpha-1,i]+2)|\ j<k \ \&\ (i,j)\in V_{\alpha-1}\}$, if D_α has two dominators.

We maintain a DP table of size $O(n)$. Each entry $DP[k]$ of this table corresponds to a point p_k that lies in the block D_α, and it indicates $\mathcal{A}[\alpha,k]$. During the execution while using $\mathcal{A}[a,k]$, we first check $D[k]$ for a non-zero entry. If it is not available, the recursive procedure for computing $\mathcal{A}[\alpha,k]$ is invoked, and on return the computed value of $\mathcal{A}[\alpha,k]$ is stored in $D[k]$. Disregarding the recursive call, computation of each $\mathcal{A}[\alpha,k]$ involves computing $\min_{i\in D_{\alpha-1}} \mathcal{A}[\alpha-1,i]$ such that there exists at least one $j\in D_\alpha$ where $(i,j)\in V_{\alpha-1}$. This needs inspecting $\mathcal{A}[\alpha-1,i]$ for at most $O(D_{\alpha-1})$ entries in the table DP. If we consider all possible $\{(i,j)|i\in D_{\alpha-1}, j\in D_\alpha\}$, we may require to spend $O(|D_\alpha|\times|D_{\alpha-1}|)$ time. In order to avoid that, we use the two pointers L and H (see Fig. 1) to mark the relevant entries in $D_{\alpha-1}$ that require to be considered for computing $\mathcal{A}[\alpha,k]$, and then in a sequential scan among the entries in $D_{\alpha-1}$ we consider the $\mathcal{A}[\alpha-1,i]$ values for the marked elements to compute their minimum.

- Fix L at the right-most element of $D_{\alpha-1}$. Let ℓ be the left-most element of D_α and r be the right-most element of $D_{\alpha-1}$.
- We start by setting $j=\ell$ (as the left dominator in D_α). We set (i) $L=r$ and (ii) H by the left-most element in $D_{\alpha-1}$ such that $d(r,j)\geq d(r,H)$. All the elements in $D_{\alpha-1}$ lying between L and H are marked as the possible dominators when p_j is a dominator in D_α.
- In each subsequent step, we set $j=j+1$. Let β be the leftmost element in $D_{\alpha-1}$ such that $d(r,j)\geq d(r,\beta)$ (β is observed by inspecting every element of D_α starting from L). If $\beta<H$, then L is set to H; otherwise L is set to β. Next, H is updated as the leftmost element in $D_{\alpha-1}$ such that $d(r,j)\geq d(r,H)$ (H is also observed by inspecting every element of D_α starting from the value of H before its updating).
- Process continues until $J=k$ is reached.

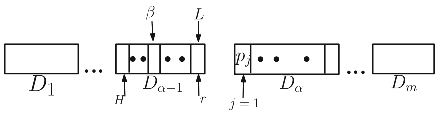


Fig. 1. Illustrating the pointers L and R used in the DP table.

From the above discussion, we have the following theorem:

Theorem 2. *Given a set $P = R \cup B$ of bi-colored points on a real line, the* `mcs` *of the point set P can be computed in $O(n^2)$-time using $O(n)$ space.*

Proof. Follows from the fact that, (i) each element of DP table is filled only once, (ii) while using $\mathcal{A}[\alpha, k]$, if it is observed to be zero, it can be set by observing $\mathcal{A}[\alpha - 1, i]$ for at most $|D_{\alpha-1}|$ marked elements of DP table, and (ii) for marking the necessary elements for all possible elements j of D_α, both L and R visits at most $|D_{\alpha-1}|$ elements of the array $\mathcal{A}$. The space complexity follows from the fact that the size of the DP table is $O(n)$. □

Recall that, earlier in Lemma 1 we have proved that each monochromatic block is self-dominating in any feasible solution. Hence for k-colored points on a real line, we conclude that the exact and the appoximation algorithm is extendable.

Corollary 1. *Given k colored points on a real line*

- *There exists and $O(n \log n)$-time 2-factor approximation algorithm for k-MCS-$\mathbb{R}$.*
- *There exists an $O(n^2)$-time using $O(n)$-space algorithm for k-MCS-$\mathbb{R}$.*

2-*MCS Problem on a Circle:* We begin with the following observation.

Observation 1. *Let $P = R \cup B$ be a set of points on the circumference of a circle C, where the points in the set R (resp. B) are colored red (resp. blue). In the nearest neighbor Voronoi diagram of P, the Voronoi edges merge at the center of C.*

Based on Observation 1, we know that for any three consecutive monochromatic blocks (D_{i-1}, D_i, D_{i+1}), in the nearest neighbor Voronoi diagram of P, no point from D_i shares boundary with the points from other blocks except D_{i-1}/D_{i+1}. This gives us an ordering (clockwise/counterclockwise) of the blocks. Thereby by using the algorithm of Sect. 2.1, we conclude the following theorem.

Theorem 3. *Let $P = R \cup B$ be a set of points on the circumference of a circle C, where the points in the set R (resp. B) are colored red (resp. blue). The* `mcs` *of P can be computed in $O(n^2)$-time using $O(n)$-space.*

***Online algorithm for* 2-*MCS*-($\mathbb{R}$):** Here, we consider the online version of the 2-`mcs` problem where the points are inserted in sequential order on a real line $\mathcal{L}$. Let us consider $P = R \cup B$ be the initial set of bicolored points on $\mathcal{L}$.

Note that, initially P can be empty. However, in that case for the first point we are not required to compute anything as the point itself is the solution. Now let Q be set of bicolored points that are going to be introduced sequentially. We denote $Q_\ell \subseteq Q$ as the set of points that are already inserted until level ℓ (i.e., ℓ-th iteration). Let P' be the `mcs` of P that is computed optimally by using the algorithm in Sect. 2.1. Clearly, at the level ℓ the optimal `mcs` can be computed in $(|P \cup Q_\ell|)^2$ time. Moreover, we show that at any level, it is unnecessary to consider all points. Let $D_1, \ldots, D_m$ are the monochromatic blocks listed by visiting the points of $(P \cup Q_\ell)$ from left to right. At level $\ell + 1$, when a new point is introduced it can affect at most three blocks. For detailed description; see Fig. 2. Thereby we conclude the following theorem.

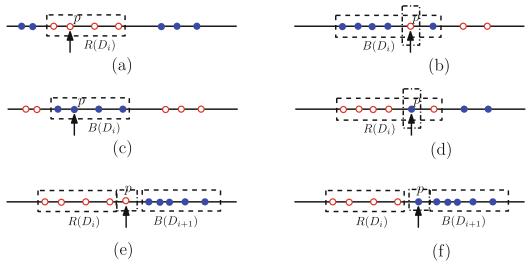


Fig. 2. Illustration of the Cases while a point p is inserted. (Color figure online)

Theorem 4. $[\star]$[2] *Given a set P of bi-colored points on a real line $\mathcal{L}$ with its* `mcs` *$P' \subseteq P$, and let Q be the set of new bi-colored points that are introduced sequentially. In each level, our algorithm recomputes the* `mcs` *in $O(r^2)$ time, where r is the max of sum of lengths of any three consecutive blocks.*

2.2 PTAS for k-MSS($\mathbb{R}^2$):

Here, we give a PTAS for the k-MSS($\mathbb{R}^2$) by adopting the PTAS for the Minimum Hitting Set (MHS) problem for a set of disks given by Mustafa et al. [10].

Minimum Hitting Set in $\mathbb{R}^2$ (MHS-$\mathbb{R}^2$)
Input: A set P of n points and a set $\mathcal{C}$ of m circular disks in $\mathbb{R}^2$
Task: Compute the smallest subset $Y \subseteq P$ such that each $C \in \mathcal{C}$ contains some point from P.

Theorem 5. [10] *MHS-$\mathbb{R}^2$ has a PTAS which runs in time $m \cdot n^{O(\epsilon^{-2})}$*

[2] We omit the proofs of results marked with $[\star]$ here due to space considerations. The proofs will be included in the longer version.

Consider any two points p, q, let $C_p(q)$ denote the disk centered at p and whose radius is the distance $d(p, q)$ between p and q. The next lemma shows that for any mss the set P'_i is a hitting set for a particular set of disks.

Lemma 3. *Let H be a* mss *for the point set $P_1 \cup P_2 \cup \ldots \cup P_k$. Let $H_i = H \cap P_i$. Then H_i is a hitting set for the instance of MHS-$\mathbb{R}^2$ whose point set is P_i and set of disks is given by $\{C_p(q) \ : \ p \in P_i, q \notin P_i\}$.*

Proof. For the sake of contradiction, assume there is some $p \in P_i$ and $q \notin P_i$ such that $C_p(q)$ contains no point of H_i that implies the nearest neighbor of p in $\left(\bigcup_{j=1, j\neq i}^{k} P_j\right) \cup H_i$ is not from H_i. This contradicts the definition of a mss. □

Algorithm 1. PTAS for k-MSS($\mathbb{R}^2$)
Input : k sets $P_1, P_2, \ldots, P_k$ of points in $\mathbb{R}^2$

```
for each $1 \le i \le k$ do
    Use Theorem 5 to obtain an $(1+\epsilon)$-approximation, say $H'_i$, for the instance
    of MHS-$\mathbb{R}^2$ where the point set is $P_i$ and the set of disks is given by
    $\mathcal{C}_i = \{C_p(q) \ : \ p \in P_i, q \notin P_i\}$
Output: $\bigcup_{i=1}^{k} H'_i$
```

Now by using Lemma 3 we give the PTAS for k-MSS($\mathbb{R}^2$) and conclude this section with the following theorem.

Theorem 6. *Algorithm 1 runs in time $n^{O(\epsilon^{-2})}$ and gives a $(1+\epsilon)$-approximation for k-MSS($\mathbb{R}^2$).*

Proof. First we analyze the running time. For each $i \in [k]$, we run the PTAS of Mustafa and Ray [10] on an instance with $|P_i| \le n$ points and the number of disks is $|\mathcal{C}_i| = |P_i| \cdot |\bigcup_{j=1, j\neq i}^{k}| \le n^2$. Hence, the total running time is $k \cdot |\mathcal{C}_i| \cdot |P_i|^{O(\epsilon^{-2})} \le n \cdot n^2 \cdot n^{O(\epsilon^{-2})} = n^{O(\epsilon^{-2})}$. Let H be an optimal mss for the k-MSS($\mathbb{R}^2$) instance. By Lemma 3, for each $i \in [k]$ we know that $H \cap P_i$ is a hitting set for the instance of MHS-$\mathbb{R}^2$ whose point set is P_i and set of disks is given by $\{C_p(q) \ : \ p \in P_i, q \notin P_i\}$. By Theorem 5, for each $i \in [k]$, the set H'_i gives a $(1+\epsilon)$-approximation for $H \cap P_i$, i.e., $|H'_i| \le (1+\epsilon) \cdot |H \cap P_i|$. Therefore, we have that $|\bigcup_{i=1}^{k} H'_i| = \sum_{i=1}^{k} |H'_i| \le \sum_{i=1}^{k} (1+\epsilon) \cdot |H \cap P_i| = (1+\epsilon) \cdot \sum_{i=1}^{k} |H \cap P_i| = (1+\epsilon) \cdot |H|$.

3 Hardness Results

3.1 NP-Hardness for MCS and MSS in Graphs

Consider a graph $G = (V, E)$ where $V = V_R \cup V_B$. The vertices $v_i \in V_R$ are colored red and $v_j \in V_B$ are colored blue. The edge length between two adjacent vertices $(v_i, v_j) = 1$ for all $v_i, v_j \in V$. The distance is measured across the edges of the graph; precisely the distance between two vertices v_i and v_j in the graph G is the length of the shortest path from v_i to v_j in the graph. Our task is to find a subset of vertices $V'_R \subseteq V_R$ and $V'_B \subseteq V_B$ (where V'_R and V'_B acts as the facility sets of red and blue vertices respectively) such that for any vertex $v_r \in V_R \setminus V'_R$, the nearest neighbor of v_r among the point set $V'_R \cup V'_B$ is a point $v'_r \in V'_R$ i.e. $N(v_r, V'_R \cup V'_B) = v'_r \in V'_R$. Similarly for any point $v_b \in V_B \setminus V'_B$, $N(v_b, V'_R \cup V'_B) = v'_b$, $v'_b \in V'_B$. Our objective is to *minimize* $|V'_R \cup V'_B|$. As defined earlier when the domain $\mathcal{X}$ is a graph then we call the problems simply as k-MCS and k-MSS.

We will show here that k-MCS and k-MSS are NP-hard.

Our reduction is from dominating set problem on a graph $G = (V, E)$. From G, we construct another graph $G' = G_1 \cup G_2 \cup v_{sp}$ where G_1 and G_2 are the copies of G and v_{sp} is a singleton node; see Fig. 3. Each edge of G_1 and G_2 has weight 1. Every vertices of G_1 are directly connected by edges with all the vertices of G_2 with weights of edge $2 - 3\epsilon$. Additionally, all vertices of G_2 are directly connected by edges of weight ϵ to v_{sp}. All the vertices of G_1 are colored red and all the vertices of G_2 are colored blue and v_{sp} is also colored blue. This completes the construction.

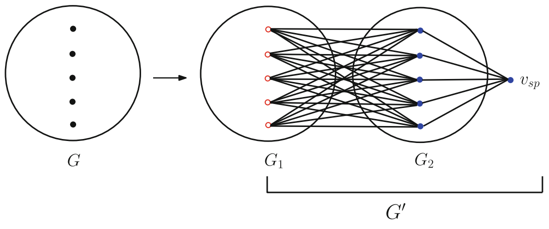


Fig. 3. The edges of graph G are not shown to keep it clear. Proof of Theorem 7 (Color figure online)

We now have the following lemma.

Lemma 4. [$\star$] *G has a dominating set of size k if and only if G' has a consistent subset of size $k + 1$.*

Further note that 2-MCS is W[2]-hard parameterized by the size k of the MCS we are looking for since Dominating Set is W[2]-hard parameterized by the size of the solution, and the parameter blowup in the reduction above is linear. Hence, the $f(k) \cdot n^{o(k)}$ lower-bound for Dominating Set under ETH [1, Theorem 5.3] also translates to 2-MCS. This shows that the brute force algorithm which runs in time $\binom{n}{k} \cdot n^{O(1)} = n^{O(k)}$ is essentially optimal. From the above discussion, we conclude the following theorem.

Theorem 7. *2-MCS on graph is NP-hard. Moreover, 2-MCS on graph is W[2]-hard parameterized by the size k of the MCS we are looking for, and under ETH there is no $f(k) \cdot n^{o(k)}$ algorithm for checking if the given instance of 2-MCS has the solution size $\leq k$.*

In fact, the same proof shows that 2-MSS is also NP-hard. If we pick a dominating set from the red vertices, then each red vertex is at a distance of ≤ 1 from a red facility. Two blue vertices are at a distance of at most 2ϵ from each other, while the distance between any blue vertex and any red vertex is at least $2 - 3\epsilon$. Hence, the consistent subsets chosen above are actually selective subsets. This implies the following result:

Corollary 2. *2-MSS is NP-hard.*

3.2 2-MSS-$\mathbb{R}^2$ is Contained in W[1]

In this section, we consider the mss problem when the points are on $\mathbb{R}^2$. We show that 2-MSS-$\mathbb{R}^2$ is contained in W[1]. The proof is by providing a non-deterministic algorithm that has an FPT time with deterministic preprocessing, then a nondeterministic phase where the number of steps is only dependent on the parameter. We will use the following result.

Theorem 8. [11] *A parameterized problem is in W[1] if and only if it can be computed by a nondeterministic RAM program accepting the input that*

1. *performs at most $f(k)p(n)$ deterministic steps;*
2. *uses at most $f(k)p(n)$ registers;*
3. *contains numbers smaller than $f(k)p(n)$ in any register at any time;*
4. *for any run on any input, the nondeterministic steps are among the last $g(k)$ steps.*

Here n is the size of the input, k is the parameter, p is a polynomial and f, g are computable functions. The non-deterministic instruction is defined as guessing a natural number between 0 and the value stored in the first register, and storing it in the first register. Acceptance of an input is defined as having a computation path that accepts.

Theorem 9. *2-MSS-$\mathbb{R}^2$ is contained in $W[1]$.*

Proof. Let $P = R \cup B$ be the set of points. For each blue point $b_i \in B$, we compute all possible distances to each red point $r_i \in R$ and store in an array A_{RB}. Similarly, we do for each red point. We will discuss for one color class here, the other follows similarly. Note that $|A_{RB}| \leq n^2$. For each value $d_i \in A_{RB}$, let $\mathcal{C}_{d_i}(B)$ be the set of disks of radius d_i centered at the points of B. For each $\mathcal{C}_{d_i}(B)$, we construct an arrangement of the disks known as vertical decomposition by projecting (ray shooting) a vertical ray up and down from each of the $O(k^2)$ intersection points between the disks, and also from the left-most and right-most point of each circle. Each ray is continued until it hits a disk or infinity

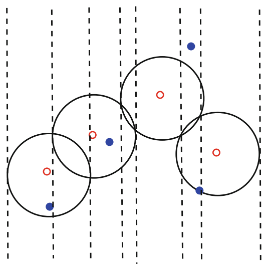


Fig. 4. Arrangement of the disks. Illustrating Theorem 9. (Color figure online)

(See Fig. 4). Note that each face is defined by at most 4 disks. The total number of vertices, edges and faces is of $O(k^2)$ [9].

For each $d_i \in A_{RB}$, first we compute all possible faces of a vertical decomposition of any subset $B' \subseteq B$ of size at most k by looking at all possible combinations of disks taking 4 at a time from $\mathcal{C}_{d_i}(B)$ (disks of radius d_i centered at the points of B). We maintain a look up table in $O(n^4)$ time that contains the total number of input points covered by each faces, The rest of the algorithm deterministically checks if B' is 2-MSS. In order to do that, for each $d_i \in A_{RB}$, we first compute a vertical decomposition of $\mathcal{C}_{d_i}(B')$ in $O(k^2)$ time. One can retrieve the number of input points from each of the $O(k^2)$ resulting faces of $\bigcup \mathcal{C}_{d_i}(B')$ from the look up table in constant time. Finally, we accept if these k points satisfy the conditions of the 2-MSS which can be deterministically checked from the intersection graph of the disks centered at the blue points. By Theorem 8 we conclude that 2-MSS of the input point set P is in $W[1]$. $\square$

3.3 Streaming Lower Bounds

In this section, we consider the streaming model: the points arrive in some order, and we need to either *store* a point or forget about it. Our goal is to show a lower bound on the storage of any streaming algorithm that solves the 2-MCS-($\mathbb{R}$) and the 2-MSS-($\mathbb{R}$) problem. We reduce from the Disjointness problem, for which there is a known lower bound of $\Omega(n)$ bits for one-way communication:

Disjointness
Input: Alice has a binary string $X \in \{0,1\}^N$, and Bob has a binary string $Y \in \{0,1\}^N$
Question: Bob wants to check if there exists an index $i \in [N]$ such that $X_i = 1 = Y_i$?

We conclude this section with the following theorem.

Theorem 10. *Any (randomized)*[3] *streaming algorithm for either the 2-MCS-($\mathbb{R}$) problem or the 2-MSS-($\mathbb{R}$) problem needs to store at least $\Omega(\sqrt{n})$ bits.*

[3] Succeeding with probability at least 2/3.

Proof. Alice performs the following action given her string $X = X_1X_2 \ldots X_i \ldots X_N$: for each $1 \leq i \leq N$

- If $X_i = 0$ then Alice does not add any points
- If $X_i = 1$ then Alice adds $N+1$ blue points, one at each of the co-ordinates $i + \frac{2\ell+1}{4(N+1)}$ for each $0 \leq \ell \leq N$

Bob performs the following action given his string $Y = Y_1Y_2 \ldots Y_j \ldots Y_N$: for each $1 \leq j \leq N$

- If $Y_j = 0$ then Bob does not add any points
- If $Y_j = 1$ then Bob adds $N+1$ red points, one at each of the co-ordinates $j + \frac{2\ell}{4(N+1)}$ for each $1 \leq \ell \leq N+1$

We now show that Disjointness answers YES if and only if size of the `mcs` for the 2-MCS($\mathbb{R}$) instance is at least $N+1$. Fix $i \in [N]$. We have four cases:

- If $X_i = 0 = Y_i = 0$ then we do not add any points
- If $X_i = 0$ and $Y_i = 1$ then we add a block of $N+1$ red points. From these set of red points, it is enough to include at most one of these red points in any consistent subset
- If $X_i = 1$ and $Y_i = 0$ then we add a block of $N+1$ blue points. From these set of blue points it is enough to include at most one of these blue facilities in any consistent subset.
- If $X_i = 1 = Y_i$ then we add $N+1$ blue points and $N+1$ red points. Moreover, these points are interleaved. It is easy to see that if there is a blue vertex interleaved between two red vertices, then this blue vertex has to be part of every consistent subset. Hence, we need to pick at least $N+1$ vertices from this set of $2N+2$ points.

Hence if Disjointness answers YES then the size of the `mcs` is $\geq N+1$, and otherwise the size of the `mcs` is at most N. There is a lower bound of $\Omega(n)$ bits of communication between Alice and Bob to solve the Disjointness problem, even allowing randomization[4] [2]. Since the number of points we introduced can be as large as $n = O(N^2)$, the lower bound of $\Omega(N)$ translates to $\Omega(\sqrt{n})$. It is easy to see that exactly the same reduction also works for 2-MSS($\mathbb{R}$). $\square$

4 Conclusions and Open Problems

In 1992, Wilfong [3] introduced the k-MCS-($\mathcal{X}$) and the k-MSS-($\mathcal{X}$) problems, and proved the NP-completeness of 3-MCS-($\mathbb{R}^2$) and 2-MSS-($\mathbb{R}^2$). Since then several heuristics have been proposed for these problems, as they have several applications in computer vision and machine learning. In this paper we renewed the theoretical study of these two problems and obtained several algorithmic and hardness results through various paradigms such as polytime algorithms, NP-hardness, parameterized complexity, streaming algorithms, etc. We hope that

[4] We require the algorithm to be correct with probability 2/3.

this work leads to more theoretical analysis of the two problems of k-MCS-($\mathcal{X}$) and the k-MSS-($\mathcal{X}$) in practically relevant scenarios. There are several interesting questions such as designing approximation or FPT algorithms for k-MCS-($\mathcal{X}$) and k-MSS-($\mathcal{X}$) on graphs or special geometric settings. Finally, we restate the question of Wilfong [3] about the complexity of 2-MCS-($\mathbb{R}^2$): is it solvable in polynomial time, or NP-complete?

References

1. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the exponential time hypothesis. Bull. EATCS **105**, 41–72 (2011)
2. Kushilevitz, E., Nisan, N.: Communication Compelxity. Cambridge University Press, Cambridge (1997)
3. Wilfong, G.T.: Nearest neighbor problems. Int. J. Comput. Geom. Appl. **2**(4), 383–416 (1992)
4. Levinson, S.E.: Structural methods in automated speech recognition. Proc. IEEE **73**(11), 1625–1650 (1985)
5. Hart, P.E.: The condensed nearest neighbor rule. IEEE Trans. Inf. Theory **14**(3), 515–516 (1968)
6. Tappert, C.C., Suen, C.Y., Wakahara, T.: The state of the art in online handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **12**(8), 787–808 (1990)
7. Gates, G.W.: The reduced nearest neighbour rule. IEEE Trans. Inf. Theory **18**(3), 431–433 (1972)
8. Masuyama, S., Ibaraki, T., Hasegawa, T.: The computational complexity of the m-center problems in the plane. IEEE Trans. IECE Jpn. **64**(2), 57–64 (1981)
9. Agarwal, P., Pach, J., Sharir, M.: State of the union-of geometric objects. In: Godman, J., Pach, J., Pollack, R. (eds.) Surveys in Discrete and Computational Geometry Twenty Years Later. Contemporary Mathematics, vol. 453, pp. 9–48 (2008)
10. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problem. In: Proceedings of the 27th(ACM) Symposium on Computational Geometry, pp. 17–22 (2009)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory, Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-29953-X
12. Hitter, G.L., Woodruff, H.B., Lowry, S.R., Isenhour, T.L.: An algorithm for a selective nearest neighbor rule. IEEE Trans. Inf. Theory **21**, 665–669 (1975)
13. Agarwal, P.K., Sharir, M.: Red-blue intersection detection algorithms, with applications to motion planning and collision detection. SIAM J. Comput. **19**(2), 297–321 (1990)
14. Arkin, E.M., Daz-Bez, J.M., Hurtado, F., Kumar, P., Mitchell, J.S.B., Palop, B., Prez-Lantero, P., Saumell, M., Silveira, R.I.: Bichromatic 2-center of pairs of points. Comput. Geom. **48**(2), 94–107 (2015)

A Polynomial Sized Kernel for Tracking Paths Problem

Aritra Banik[1], Pratibha Choudhary[1(✉)], Daniel Lokshtanov[2], Venkatesh Raman[3,4], and Saket Saurabh[2,3,4]

[1] Indian Institute of Technology Jodhpur, Jodhpur, India
aritrabanik@gmail.com, pratibhac247@gmail.com
[2] University of Bergen, Bergen, Norway
{daniello,saket.saurabh}@ii.uib.no
[3] The Institute of Mathematical Sciences, HBNI, Chennai, India
{vraman,saket}@imsc.res.in
[4] UMI ReLaX, Chennai, India

Abstract. Consider a secure environment (say an airport) that has a unique entry and exit point with multiple inter-crossing paths between them. We want to place (minimum number of) trackers (or check points) at some specific intersections so that based on the sequence of trackers a person has encountered, we can identify the exact path traversed by the person. Motivated by such applications, we study the Tracking Paths problem in this paper. Given an undirected graph with a source s, a destination t and a non-negative integer k, the goal is to find a set of at most k vertices, a tracking set, that intersects each s-t path in a unique sequence. Such a set enables a central controller to *track* all the paths from s to t. We first show that the problem is NP-complete. Then we show that finding a tracking set of size at most k is fixed-parameter tractable (FPT) when parameterized by the solution size. More specifically, given an undirected graph on n vertices and an integer k, we give a polynomial time algorithm that either determines that the graph cannot be tracked by k trackers or produces an equivalent instance of size $\mathcal{O}(k^7)$.

1 Introduction

Tracking moving objects in a secure environment is an active area of research. Typically a secure environment is modeled as a network with fixed entry and exit point(s). Monitoring is achieved by placing sensor nodes which monitor the movements of the objects in the network. For a detailed study of field surveillance for the purpose of habitat monitoring, securing buildings, and intruder tracking please refer to [3,6]. While tracking traces of illegal activities over Internet, the biggest challenge is to track moving data packets [13,15]. One may want to place trackers at an appropriate subset of routers in the network in order to track such activities.

Motivated by these applications, in a recent paper, Banik et al. [2] considered the problem of target tracking theoretically and modeled it as the following graph

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 94–107, 2018.
https://doi.org/10.1007/978-3-319-77404-6_8

theoretic problem. Let $G = (V, E)$ be an undirected graph without any self loop or parallel edges and suppose G has a unique entry vertex s and a unique exit vertex t. A simple path from s to t is called an s-t path. Let P be an s-t path in G and $A \subseteq V$ be a set of vertices. We denote by $\mathcal{T}_P^A$ the sequence of vertices of A obtained from P by deleting the vertices that do not belong to A. A set of vertices A is a tracking set for G, if and only if for any two distinct s-t paths P_1 and P_2, $\mathcal{T}_{P_1}^A \neq \mathcal{T}_{P_2}^A$. The vertices in set A are called trackers. Banik et al. [2] proved that the problem of finding a minimum-cardinality tracking set with respect to *shortest* s-t paths (Tracking Shortest Paths problem) is NP-hard and APX-hard. In this paper we consider the problem of tracking all simple paths and not just the shortest paths. In particular, we study the following.

Tracking Paths (G, s, t, k) **Parameter:** k
Input: An undirected graph $G = (V, E)$ with two distinguished vertices s and t, and a non-negative integer k.
Question: Is there a tracking set T of size at most k for G?

Our Results and Methods. In this paper we study Tracking Paths from the perspective of parameterized complexity. Our first contribution is the following.

Theorem 1. Tracking Paths *is* NP-*complete.*

As is the case for any proof of NP-completeness, the proof of Theorem 1 requires two steps: hardness and containment inside NP. While hardness follows from the result of Banik et al. [2] (Shown in full version of paper), it is not clear why the problem is in NP. To check whether a given set S is a tracking set for G, we need to go over all pairs of paths between s and t and check whether the sequence of trackers used on them are distinct. However, the number of paths between s and t could be exponential and thus it does not seem possible to exploit definition of the problem to show it inside NP. Thus, in order to show that the problem belongs to NP, we first give an alternate characterization for a tracking set. Then, to give a polynomial time algorithm to check whether a given set T is a tracking set, we first show that T is a *feedback vertex set* (FVS). That is, $G \setminus T$ is a forest. Using this property and our alternate characterization of tracking set we give a polynomial time algorithm to test whether a given set T is a tracking set.

Once we have shown that Tracking Paths is NP-complete, we study the problem from the viewpoint of parameterized complexity. In particular we design an FPT algorithm for Tracking Paths. That is, we design an algorithm for Tracking Paths with running time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$, where k is size of the tracking set we seek. In fact, we prove a stronger result than just designing an FPT algorithm; we give a polynomial kernel for the problem. In particular, given an instance (G, s, t, k), we give a polynomial time algorithm that either determines that (G, s, t, k) is a NO instance or produces an equivalent instance with $\mathcal{O}(k^6)$ vertices and $\mathcal{O}(k^7)$ edges. This polynomial time algorithm is called a *kernelization algorithm* and the reduced instance is called a *kernel.* For more details about parameterized complexity and kernelization we refer to monographs [4,5]. Our second contribution is the following result.

Theorem 2. Tracking Paths *admits a polynomial kernel of size* $\mathcal{O}(k^7)$.

The kernelization algorithm (proof of Theorem 2) works along following lines. Let (G, s, t, k) be an input instance to Tracking Paths. We first apply a simple reduction rule to ensure that each edge and vertex in G belongs to some path from s to t. Then we prove two structural claims: (a) every tracking set S is an FVS; and (b) if G has a pair of vertices x and y such that x and y have at least $k+4$ vertex disjoint paths between them, then (G, s, t, k) is a NO instance. Next, using the known factor 2 approximation algorithm for the Feedback Vertex set problem, we compute a set S such that $G \setminus S$ is a forest. If $|S| > 2k$ then we immediately return that (G, s, t, k) is a NO instance. Thus, assume that S is of size at most $2k$. Now using the second structural claim regarding tracking set, we show that the number of connected components of $G \setminus S$ and the number of vertices in $V(G \setminus S)$ that have at least two neighbors in S are upper bounded by $k^{\mathcal{O}(1)}$. Next, we bound the number of vertices in $V(G \setminus S)$ that have *exactly* one neighbor in S. To do this, we fix a tree R in $G \setminus S$ and a vertex $v \in S$ and bound the size of the set of neighbors of v, $N_R(v)$, in R. Towards this, we consider the minimal subtree T in R that contains all the neighbors of v, and show that if $|N_R(v)| > k^{\mathcal{O}(1)}$, then we can partition the tree T into $k+1$ parts in such a way that each part must contain a tracker. This bounds the degree of each vertex in S into $V(G \setminus S)$ by $k^{\mathcal{O}(1)}$. This together with well-known counting methods on trees gives us the desired polynomial kernel for Tracking Paths.

Related Work. Different structural properties of a graph have been studied previously to analyze navigational models in network setting. In a seminal paper, Slater [14] introduced the concept of metric dimension of a graph. In graph theory, the metric dimension of a graph G is the minimum cardinality of a subset S of vertices such that all other vertices are uniquely determined by their distances to the vertices in S [7]. One application of metric dimension is the problem of determining the location of an object in a network depending on its distance from different landmarks in the network [11]. For a survey of metric dimension in graphs see [8]. Furthermore, as mentioned before Tracking Paths is also closely related to Feedback Vertex set in a graph. Feedback Vertex set is NP-hard [9], has a 2 approximation algorithm [1], a quadratic sized kernel [16] and an FPT algorithm running in time $\mathcal{O}((3.619)^k n^{\mathcal{O}(1)})$ [12].

Notations. A kernelization algorithm is obtained using what are called *reduction rules*. These rules transform the given parameterized instance in polynomial time to another equivalent instance, and a rule is said to be *safe* if the resulting graph has a tracking set of size at most k if and only if the original instance has one. For a path P, $V(P)$ denotes the vertex set of path P, and for a subgraph G', $V(G')$ denotes the vertex set of G'. Let P_1 be a path between vertices a and b, and P_2 be a path between vertices b and c, such that $V(P_1) \cap V(P_2) = \{b\}$. By $P_1 \cdot P_2$, we denote the path from a to c created by concatenating P_1 and P_2.

2 NP-Completeness

In this section we show that Tracking Paths is NP-complete. We first show that the problem is NP-hard. Towards that we will use the result of Banik et al. [2] which showed that Tracking Shortest Paths in a graph is NP hard by giving a reduction from Vertex Cover. We show that the same reduction also proves that the problem is NP hard when we want to track all s-t paths. In particular we prove the following hardness result in the full version of the paper.

Lemma 1. $\circledast$[1] *For any graph $G = (V, E)$ we can construct a graph G' on $|V| + |E| + 5$ vertices such that, there exists a vertex cover of size k for G if and only if there exists a tracking set of size $k + |E| + 1$ for G'.*

In what follows, we show that the problem is NP-complete. Towards that we first give an alternate characterization for a tracking set. Then using a pre-processing rule, we show that for a graph, every tracking set is also a feedback vertex set (FVS). Finally using these two properties we devise a polynomial time algorithm to check whether a given set of vertices is a tracking set for graph G.

2.1 Characterization of Tracking Set

Towards characterization of tracking set, we first define tracking set condition. For a graph $G = (V, E)$ a set of vertices $V' \subseteq V$ is said to satisfy tracking set condition if there does not exist a pair of vertices $u, v \in V$, such that there exists two distinct paths, say P_1, P_2, between u and v in $G \setminus \{V' \cup \{s, t\}\} \cup \{u, v\}$, and, there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt} in $G \setminus \{V(P_1) \cup V(P_2)\} \cup \{u, v\}$, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$. See Fig. 1.

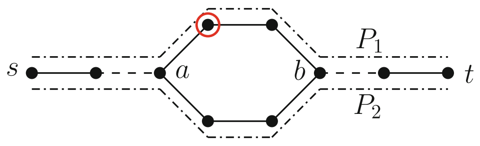


Fig. 1. Graph satisfying tracking set condition

Thus we have the following lemma.

Lemma 2. *For a graph $G = (V, E)$, a set of vertices $T \subseteq V$ is a tracking set if and only if T satisfies the* tracking set condition.

Proof. Let us assume that $T \subseteq V$ is a tracking set for G. We claim that T satisfies the tracking set condition. Suppose not. Then there exists a pair of vertices $a, b \in V$, such that there exists two distinct paths, say P', P'', between a and b in $G \setminus \{T \cup \{s, t\}\} \cup \{a, b\}$, and, there exists a path from s to a, say P_{sa}, and a path

[1] Proofs for lemmas/corollaries marked with $\circledast$ will appear in full version of the paper.

from b to t, say P_{bt} in $G \setminus \{V(P') \cup V(P'')\} \cup \{a, b\}$, such that $V(P_{sa}) \cap V(P_{bt}) = \emptyset$. Notice that there might be trackers in $V(P_{sa}) \cup V(P_{bt})$, however there are no trackers in $V(P') \cup V(P'') \setminus \{a, b\}$. Observe now there exists two s-t paths in G, $P_1 = P_{sa} \cdot P' \cdot P_{bt}$ and $P_2 = P_{sa} \cdot P'' \cdot P_{bt}$ that have the same sequence of trackers. This contradicts the assumption that T is a tracking set for G.

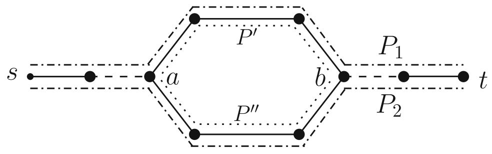


Fig. 2. Graph not satisfying tracking set condition

Conversely, let us assume that $T \subseteq V$ satisfies the tracking set condition. We claim that T is a tracking set for G. Suppose not. Then there exists at least two s-t paths in G, say P_1 and P_2 that contain the same sequence of trackers. Hence, $V(P_1) \setminus V(P_2) \cup V(P_2) \setminus V(P_1)$ must not contain any tracker. Notice that s is the first vertex that appears in both P_1 and P_2. Starting from s, keep scanning the vertices in P_1 and P_2 as long as the vertices in these paths are the same. Let a be the first vertex that appears in both P_1 and P_2, such that after a the next vertex appearing in P_1 is not the same as that appearing in P_2. Continue scanning the vertices in both paths, until a vertex b is found such that b appears in both P_1 and P_2. See Fig. 2. Note that there exists two distinct paths between a and b in $G \setminus \{T \cup \{s, t\}\} \cup \{a, b\}$. Let us call these paths P' and P''. Observe that there must exist such a vertex a and a vertex b in $V(P_1) \cup V(P_2)$ such that there exist distinct paths between them, else P_1 and P_2 would not be two different s-t paths. Notice that $V(P') \cup V(P'') \setminus \{a, b\}$ cannot contain any tracker, else P_1 and P_2 would not contain the same sequence of trackers. There exists a path from s to a, say P_{sa}, and a path from b to t, say P_{bt} in $G \setminus \{V(P') \cup V(P'')\} \cup \{a, b\}$, such that $V(P_{sa}) \cap V(P_{bt}) = \emptyset$, and both P_{sa} and P_{bt} may or may not contain any trackers. This contradicts the assumption that T satisfies the tracking set condition. □

Although we have a nice characterization for what qualifies to be a tracking set, we still cannot use it for verification purpose because there can be exponentially many paths between s and t. However, we show in the next subsection that in our case we can assume that between any two vertices we have only polynomially many relevant paths in the graph once we remove all the trackers along with s and t.

2.2 Tracking Set as Feedback Vertex Set

Let (G, s, t, k) be an input instance to Tracking Paths. After applying a reduction rule that ensures that each edge and vertex in G belongs to some s-t path, we can show that every tracking set is also an FVS for the reduced graph. For a graph $G = (V, E)$, an FVS is a set of vertices $S \subseteq V$ such that $G \setminus S$ is a forest.

Reduction Rule 1. *If there exists a vertex or an edge that does not participate in any s-t path then delete it.*

Lemma 3. ⊛ *Reduction Rule 1 is safe and can be implemented in polynomial time.*

In rest of the paper we assume that each vertex and each edge participates in at least one s-t path. Next we have a lemma which establishes the connection between a tracking set and an FVS.

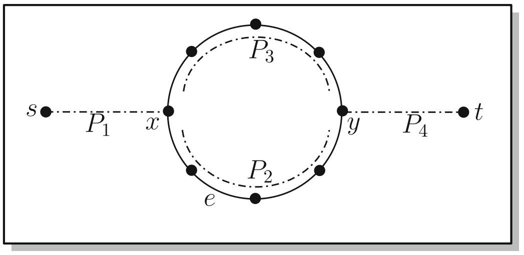


Fig. 3. Cycle without tracker

Lemma 4. *If T is a tracking set for G then T is a feedback vertex set for G as well.*

Proof. Consider any cycle C in G. We show that T contains at least one vertex from C. Consider an edge e in cycle C. Since every edge in the graph participates in at least one s-t path, let P be an s-t path that contains the edge e. Path P may contain some more vertices and edges from the cycle C. Let x be the first vertex of C that appears in P while traversing from s to t. Similarly let y be the last vertex of C that appears in path P (see Fig. 3). Observe that there are two paths between x and y in cycle C, one of them containing edge e, and the other one not containing edge e. Denote the path containing the edge e by P_2, and the other path by P_3. Let P_1 be the subpath (which would be empty if s is in the cycle) of P from s to x, and P_4 be the subpath (which would be empty if t is in the cycle) of P from y to t. Consider the following two paths:

$P' = P_1 \cdot P_2 \cdot P_4$

$P'' = P_1 \cdot P_3 \cdot P_4$

Observe that if C does not contain any tracker, then P' and P'' contain exactly the same sequence of trackers contradicting the fact that T is a tracking set. □

Thus we have the following corollary.

Corollary 1. *The size of a minimum tracking set T for G is at least the size of a minimum FVS for G.*

2.3 Verification of Tracking Set

In Lemma 4, we have shown that if T is a tracking set for G, then T is also an FVS for G. So we first check if T is an FVS for G. Using a breadth first search, in $\mathcal{O}(n+m)$ time we can check whether $G \setminus T$ is a forest or not. If not, we reject T. Henceforth, we assume that T is an FVS for G.

Lemma 5. *For a set of vertices $T \subseteq V$, we can verify in polynomial time if T satisfies the* tracking set condition.

Proof. Observe that in order to show that $T \subseteq V$ is a tracking set it is sufficient to consider the subgraph G' of G, which only has those edges and vertices that are part of some s-t path (s). That is, we first apply Reduction Rule 1 and reduce given the instance. For the clarity of presentation we denote the reduced instance also by G.

Let $G' = G \setminus \{T \cup \{s, t\}\}$. In order to verify if the tracking set condition holds for a set of vertices $T \subseteq V$, we first need to check if there exists a pair of vertices $u, v \in V$, such there exists two distinct paths between u and v in $G' \cup \{u, v\}$. We consider each pair of vertices $u, v \in V$ and check in $G' \cup \{u, v\}$ if there exists two or more distinct paths between u and v. Observe that since a tracking set is also an FVS, G' is a forest. Hence there exists a unique path between each pair of vertices in G'. If both u and v belong to G', then there exists a unique path between them, and in such a case we can skip verification of second part of tracking set condition. If either u or v, or both of them do not belong to G', then they can have at most n neighbors each in G'. Hence, the number of paths between u and v in $G' \cup \{u, v\}$ is $\mathcal{O}(n^2)$. We consider each neighbor of u and v in $G' \cup \{u, v\}$, and find the unique path between these neighbors in G', in $\mathcal{O}(n)$ time using depth first search. Thus finding all paths between u and v can be done in $\mathcal{O}(n^3)$ time. For each pair of paths, say P_1 and P_2, among the $\mathcal{O}(n^2)$ paths in $G' \cup \{u, v\}$, we verify second part of tracking set condition, i.e., we check if there exists a path from s to u, say P_{su}, and a path from v to t, say P_{vt} in $G \setminus \{V(P_1) \cup V(P_2)\} \cup \{u, v\}$, such that $V(P_{su}) \cap V(P_{vt}) = \emptyset$. This step can be performed in $\mathcal{O}(n^2)$ time using the algorithm for disjoint paths [10]. Hence the overall time taken for verification is $\mathcal{O}(n^2(n^3 + n^4 n^2))$. □

Lemmas 1 and 5 together prove Theorem 1.

3 Polynomial Kernel for Tracking Paths

In this section, with the help of some reduction rules we give a polynomial time algorithm that checks whether the given instance is a NO instance (for a solution of size at most k) or produces an equivalent instance with $\mathcal{O}(k^6)$ vertices. We assume that the given graph has been preprocessed using Reduction Rule 1.

Recall that from Corollary 1, we know that the size of a minimum tracking set T for G is at least the size of a minimum FVS for G. First we find a 2-approximate solution S in G for Feedback Vertex set using [1]. From Corollary 1, we have the following reduction rule.

Reduction Rule 2. *Apply the algorithm of* [1] *to find a 2-approximate solution, S for* FEEDBACK VERTEX SET. *If $|S| > 2k$, then return that the given instance is a NO instance.*

Observe that $\mathcal{F} = G \setminus S$ is a forest. Now we try to bound the number of vertices in graph $\mathcal{F}$ given that k trackers are sufficient to track all the s-t path in G. Towards this we first prove a monotonicity lemma and a corollary which says that if a subgraph of G cannot be tracked with k trackers, then G cannot be tracked with k trackers either.

Lemma 6. *Let $G = (V, E)$ be a graph and $G' = (V', E')$ be a subgraph of G such that $\{s, t\} \in V'$. If T is a tracking set for G and T' is a minimum tracking set for G', then $|T'| \leq |T|$.*

Proof. We show that T is a tracking set for G' as well. For otherwise, there must exist two s-t paths, say P_1 and P_2 in G' that contain the same sequence of trackers. Observe that P_1 and P_2 also belong to G. Hence, in this case P_1 and P_2 cannot be distinguished by T in G as well. This contradicts the assumption that T is a tracking set for G. Hence the lemma holds. □

Corollary 2. *If a subgraph of G that contains both s and t cannot be tracked by k trackers, then G cannot be tracked by k trackers either.*

Henceforth, we limit ourselves to analyzing the cases when a subgraph cannot be tracked by k trackers. In upcoming sections, we bound the number of vertices in $\mathcal{F}$ by a function of k. First we bound the number of vertices in $\mathcal{F}$ that have at least two neighbors in S.

3.1 Bounding the Number of Vertices in $\mathcal{F}$ with at Least Two Neighbors in S

Lemma 7. ⊛ *If there are two vertices $u, v \in V$ such that there exists more than $k + 3$ vertex disjoint paths between u and v, then G cannot be tracked with k trackers.*

From Lemma 7 we have the following.

Reduction Rule 3. *If there exists two vertices in S that have $(k + 4)$ common neighbors in $\mathcal{F}$ then we return that the given instance is a NO instance.*

Lemma 8. ⊛ *If there exists two vertices in S that have $(k + 4)$ common neighbors in $\mathcal{F}$ then G cannot be tracked with k trackers.*

Corollary 3. ⊛ *If Reduction Rule 3 is not applicable, then the number of vertices in $\mathcal{F}$ that have at least two neighbors in S is at most $\binom{2k}{2}(k+4) \leq 2k^2(k+4)$.*

3.2 Bounding the Number of Trees in $\mathcal{F}$

The argument given in the proof of Lemma 8 can be also used to bound the number of trees in $\mathcal{F}$ that are adjacent to at least two vertices in S.

Reduction Rule 4. *If there exists two vertices in S that are common neighbors to $(k+4)$ trees in $\mathcal{F}$, we return that the given instance is a NO instance.*

Lemma 9. ⊛ *If there are $(k+4)$ trees that are adjacent to two vertices u and v in S, then G cannot be tracked with k trackers.*

Corollary 4. *If Reduction Rule 4 is not applicable, then the number of trees in $\mathcal{F}$ that have at least two neighbors in S is at most $\binom{2k}{2}(k+4)$.*

Next we bound the number of trees with exactly one neighbor in S. Towards this we first show the following.

Lemma 10. *Any induced subgraph G' of G containing at least one edge will contain a pair of vertices u, v such that there exists a path in G from s to u and another path from v to t, and these paths are mutually vertex disjoint and also they do not contain any other vertices from G' except u and v.*

Proof. Consider the induced subgraph G'. Pick any edge $e = (x, y)$ of G'. We know that e participates in at least one s-t path, say P. Denote the subpath of P from s to x and from y to t by P_s and P_t respectively. Observe that P_s and P_t are vertex disjoint. Let $u \in G'$ be the first vertex in P_s while traversing from s to x and $v \in G'$ be the last vertex while traversing P_t from t to y. Observe that the subpath of P_s from s to u and the subpath of P_t from v to t are vertex disjoint and intersect with G' only at u and v. Therefore the claim holds. □

We show (in full version of the paper) that the only possible trees having exactly one neighbor in S are ones that contain s or t.

Lemma 11. *The number of trees in $\mathcal{F}$ that have a single neighbor in S is at most two.*

Proof. Let R be a tree in $\mathcal{F}$ that does not contain s or t. If R contains only one vertex, then it will have at least two neighbors in S, as due to Reduction Rule 1, G has no vertex with degree at most one or multiple edges. Hence R contains at least two vertices and hence an edge. By Lemma 10, there exists a pair of vertices u, v in R such that there is a path in G from s to u and a path from v to t that are mutually disjoint and contain no other vertex of R. This means that R has at least two neighbors outside R (the neighbors of u and v in those paths) and hence in S. So the only possible trees having a single neighbor in S are those that contain s or t, and hence the lemma follows. □

From the above two lemmas we know that every tree has at least one neighbor in S. Hence we have the following corollary.

Corollary 5. *If the Reduction Rules 1 to 4 are not applicable, then the number of trees in $\mathcal{F}$ is at most $\binom{2k}{2}(k+4) + 2 \leq 2k^2(k+4)$.*

3.3 Bounding the Number of Vertices in $\mathcal{F}$ with Exactly One Neighbor in S

Here we bound the number of vertices in $\mathcal{F}$ that have a single neighbor in S and we do that by bounding the number of vertices from a single tree in $\mathcal{F}$ that are adjacent to a particular vertex of S.

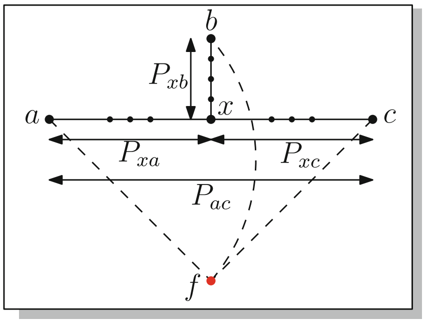


Fig. 4. Illustration of Lemma 12

Consider any tree $R \in \mathcal{F}$. Let V_f be the set of vertices of R that is adjacent to a particular vertex $f \in S$. In this section we prove a bound on the cardinality of V_f. We denote *the smallest connected subtree of* R *which contains all of* V_f *by* R'. Note that the leaf nodes of R' are in V_f. Consider three vertices $\{a, b, c\} \subset V_f$. Let R'' be a minimum subtree of R' that contains a, b and c.

Lemma 12. *Any tracking set* T *must contain at least one vertex of* R''.

Proof. Let T be a tracking set which does not contain any vertex from R''. Observe that all the leaf nodes of R'' are either a, b or c. Furthermore, at least two of a, b or c must be the leaves of R''. Without loss of generality assume that a and c are leaves of R''. Consider the path P_{ac} from a to c in R''. Without loss of generality assume that b is connected to P_{ac} via the path P_{xb} (see Fig. 4) joining P_{ac} at vertex x. We denote the paths from x to a and c by P_{xa} and P_{xc} respectively. Note that P_{xb} could be a single vertex, i.e., $b = x$.

Let G' be the graph induced by $\{f\} \cup V(R'')$. From Lemma 10 we know that there exists a pair of vertices $u, v \in V(G')$, such that there are two vertex disjoint paths in G from s to u and from v to t and both these paths do not contain any vertex from G' except u and v. We use P_s to denote the path from s to u, and P_t to denote the path from v to t (see Fig. 5(a)). Observe that depending on the locations of u and v there can be three cases.

Case 1 (u and v are in different chains among P_{xa}, P_{xb} and P_{xc}): Without loss of generality assume $u \in P_{xa}, v \in P_{xc}$, other cases can be proved similarly. Denote the paths $P_{ua} \subset P_{xa}$, $P_{xv} \subset P_{xc}$ and $P_{cv} \subset P_{xc}$ by λ_1, λ_2 and λ_3 respectively (see Fig. 5(a)). Observe that $P_s \cdot \lambda_1 \cdot f \cdot P_{xb} \cdot \lambda_2 \cdot P_t$ and $P_s \cdot \lambda_1 \cdot f \cdot \lambda_3 \cdot P_t$ contain

the same sequence of trackers (note that f may or may not contain a tracker). Hence these two paths are indistinguishable which contradicts the fact that T is a tracking set.
Case 2 (u and v are in same chain among P_{xa}, P_{xb} and P_{xc}): Without loss of generality assume $u, v \in P_{xa}$. Denote the paths $P_{ua} \subset P_{xa}$, $P_{cv} \subset P_{ac}$ and $P_{xv} \subset P_{xc}$ by λ_1, λ_2 and λ_3 respectively (see Fig. 5(b)). Observe that $P_s \cdot \lambda_1 \cdot f \cdot \lambda_2 \cdot P_t$ and $P_s \cdot \lambda_1 \cdot f \cdot P_{xb} \cdot \lambda_3 \cdot P_t$ contain the same sequence of trackers. This contradicts that T is a tracking set.
Case 3 ($u = f$ or $v = f$): Without loss of generality assume $u = f$. Proof follows from the fact that from f there exists two paths to any other vertex in R''. In particular, $P_s P_{av} P_t$ and $P_s P_{bv} P_t$ are two paths that cannot be distinguished.

This completes the proof. □

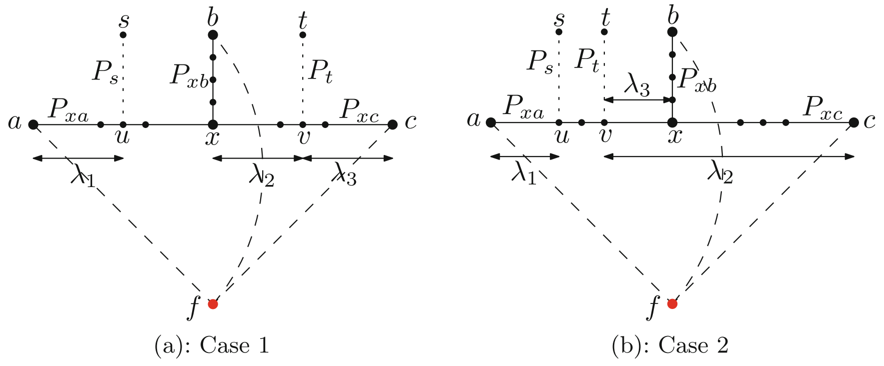


Fig. 5. Illustration of Lemma 12

Next we show the following.

Lemma 13. *The degree of each vertex in R' is at most $k + 4$.*

Proof. Observe that if there exists a vertex v of degree $k + 4$ in R', then there are at least $k + 4$ disjoint paths between v and $k + 4$ leaves of R'. As every leaf node is connected with f, there will be more than $k + 4$ disjoint paths between v and f. In that case from Lemma 7 we know that G cannot be tracked with k trackers. Therefore we have a contradiction, and hence the result holds. □

Let the vertices in R' that are adjacent to f be colored RED and the others be colored BLUE. Since R' is a minimum tree containing V_f, we have that all the leaf nodes of R' are RED. Next we have following lemma.

Lemma 14. *If the number of* RED *vertices in R' is at least $(2k+8)(k+1)$ then we can partition R' into at least $k+1$ disjoint subtrees each containing at least three* RED *vertices.*

Proof. In R', let w be the closest node to a leaf such that the subtree rooted at w has at least three RED vertices. Since w is the closest node we have that for any of its children the subtree rooted at them has at most two RED children. Using Lemma 13, we can say this implies that the subtree rooted at w has at most $2(k+4)$ RED vertices. Now remove the subtree rooted at w, and continue. This implies that if $|V(R')| \geq (2k+8)(k+1)$, we can partition R' into at least $k+1$ disjoint subtrees each containing at least three RED vertices. □

From Lemma 12 we know that each such partition needs a tracker. Thus we have the following.

Lemma 15. *If there exists $(2k+8)(k+1)$ vertices of a tree in $\mathcal{F}$ adjacent to a single vertex in S, then G cannot be tracked with k trackers.*

Reduction Rule 5. *If there is a vertex in S adjacent to $(2k+8)(k+1)$ vertices of a tree in $\mathcal{F}$, we return that the given instance is a NO instance.*

Corollary 6. ⊛ *If the Reduction Rule 5 is not applicable, then the total number of vertices from each tree in $\mathcal{F}$ that are adjacent to a vertex in S is at most $2k(2k+8)(k+1)$.*

3.4 Wrapping Up – Polynomial Kernel and FPT Algorithm

From Corollaries 4 and 6, we have following corollary.

Corollary 7. *The number of vertices in a tree of $\mathcal{F}$ that have at least one neighbor in S is at most $2k^2(k+4) + 2k(2k+8)(k+1) = 2k(3k^2+14k+8)$.*

Next we give a reduction rule that helps bound the number of internal vertices in a tree in subsequent lemma.

Reduction Rule 6. *If there are three vertices a, b and c each of degree two, such that (a,b) and (b,c) both are edges, do the following. Delete b and introduce edge (a,c) in G.*

Lemma 16. ⊛ *Reduction Rule 6 is safe and can be implemented in polynomial time.*

Lemma 17. *The number of vertices in a tree in $\mathcal{F}$ that do not have any neighbor in S is at most $10k(3k^2+14k+8)$.*

Proof. In a tree, we denote the set of leaf nodes by V_1, the set of vertices of degree two by V_2, and the set of vertices of degree three or more by V_3. Since G is preprocessed, there cannot be any degree one vertex in G except s and t. Thus each leaf of the tree in $\mathcal{F}$ necessarily has a neighbor in S. Hence by Corollary 7, number of vertices in V_1 is bounded by $2k(3k^2+14k+8)$. Observe that both V_2 and V_3 can belong to the set of internal nodes in a tree. By standard graph theoretic properties of a tree, we know that $|V_3| \leq |V_1| - 1$. Hence the number of vertices in V_3 is bounded by $2k(3k^2+14k+8) - 1$. Due to Reduction Rule 6, we know that at most two vertices of degree two can exist in series. Observe that each vertex in V_1 and V_3 can have at most two vertices from V_2 as its immediate ancestors. Hence the number of vertices in V_2 is bound by $2(|V_1| + |V_3|)$, i.e. $|V_2| \leq 8k(3k^2+14k+8)-2$. Hence $|V_2|+|V_3| \leq 2k(3k^2+14k+8)-3$. We ignore this reduction in count by 3 from our bound. Thus the overall bound on number of vertices in $\mathcal{F}$ which do not have any neighbor in S is $10k(3k^2+14k+8)$. □

From Corollary 7 and Lemma 17, we have the following corollary.

Corollary 8. *The number of vertices in a tree in $\mathcal{F}$ is at most $12k(3k^2+14k+8)$ if none of the reduction rules are applicable.*

Proof of Theorem 2. From Corollaries 5 and 8 we can say that if none of the reduction rules are applicable, then the total number of vertices in $\mathcal{F}$ is at most $12k(3k^2+14k+8)2k^2(k+4)$ and hence the total number of vertices in G is at most $12k(3k^2+14k+8)2k^2(k+4)+2k$. Thus the total number of vertices is at most $72k^6+624k^5+1536k^4+768k^3+2k$ in G.

Since $\mathcal{F}$ is a forest, total number of edges in $\mathcal{F}$ is at most $12k(3k^2+14k+8)2k^2(k+4)-1$. From Reduction Rule 5 we know that the total number of vertices from each tree in $\mathcal{F}$ that are adjacent to a single vertex in S is at most $2k(2k+8)(k+1)$. From Corollary 5 we know that the number of trees in $\mathcal{F}$ is at most $2k^2(k+4)$. Thus the total number of edges between $\mathcal{F}$ and S is $2k(2k+8)(k+1)2k^2(k+4)2k$. The total number of edges among the vertices in S is at most $4k^2$. Thus the total number of edges in G is at most $12k(3k^2+14k+8)2k^2(k+4)-1+2k(2k+8)(k+1)2k^2(k+4)2k+4k^2 = \mathcal{O}(k^7)$. □

Theorem 3. Tracking Paths *is* FPT *when parameterized by the solution size, and the running time of the* FPT *algorithm is* $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$.

Proof. We can run through all subsets of size k of the kernel and use the verification algorithm mentioned in Lemma 5 to check if a particular subset is a tracking set for G. Using the proof of Theorem 2, we can find all subsets of size k in $\mathcal{O}(k^{6k})$ time. From Lemma 5, we can verify if a subset of k vertices is a tracking set for G in polynomial time. Thus we have a $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ time FPT algorithm for Tracking Paths. □

4 Conclusions

In this paper we have shown that the Tracking Paths problem is NP-complete. We also show the problem to be fixed-parameter tractable by proving the existence of a polynomial sized kernel. This is achieved via exploiting the connection between a feedback vertex set and tracking set. An open problem is to improve the size of the kernel and using it or otherwise to improve the running time of the FPT algorithm for the problem. Other directions to explore are to find approximation algorithms and studying the problem for directed graphs.

References

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discret. Math. **12**(3), 289–297 (1999)
2. Banik, A., Katz, M.J., Packer, E., Simakov, M.: Tracking paths. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 67–79. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_7
3. Bhatti, S., Xu, J.: Survey of target tracking protocols using wireless sensor network. In: 2009 Fifth International Conference on Wireless and Mobile Communications, pp. 110–115 (2009)
4. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms, 1st edn. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0515-9
6. Gupta, R., Das, S.R.: Tracking moving targets in a smart sensor network. In: IEEE 58th Vehicular Technology Conference, vol. 5, pp. 3035–3039 (2003)
7. Harary, F., Melter, R.A.: On the metric dimension of a graph. Ars Combin **2**(191–195), 1 (1976)
8. Hernando, C., Mora, M., Pelayo, I.M., Seara, C., Wood, D.R.: Extremal graph theory for metric dimension and diameter. Electron. J. Comb. **17**(1), R30 (2010)
9. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
10. Kawarabayashi, K., Kobayashi, Y., Reed, B.: The disjoint paths problem in quadratic time. J. Comb. Theory Ser. B **102**(2), 424–435 (2012)
11. Khuller, S., Raghavachari, B., Rosenfeld, A.: Landmarks in graphs. Discret. Appl. Math. **70**(3), 217–229 (1996)
12. Kociumaka, T., Pilipczuk, M.: Faster deterministic feedback vertex set. Inf. Process. Lett. **114**(10), 556 (2014)
13. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Comput. Surv. **39**(1), Article No. 3 (2007)
14. Slater, P.J.: Leaves of trees. Congr. Numer **14**(549–559), 37 (1975)
15. Snoeren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Kent, S.T., Strayer, W.T.: Hash-based IP traceback. SIGCOMM Comput. Commun. Rev. **31**(4), 3–14 (2001)
16. Thomassé, S.: A $4k^2$ kernel for feedback vertex set. ACM Trans. Algorithms (TALG) **6**(2), 32 (2010)

Time-Space Trade-Offs for Computing Euclidean Minimum Spanning Trees

Bahareh Banyassady[1](✉), Luis Barba[2], and Wolfgang Mulzer[1]

[1] Freie Universität Berlin, Berlin, Germany
{bahareh,mulzer}@inf.fu-berlin.de
[2] ETH Zurich, Zurich, Switzerland
luis.barba@inf.ethz.ch

Abstract. In the limited-workspace model, we assume that the input of size n lies in a random access read-only memory. The output has to be reported sequentially, and it cannot be accessed or modified. In addition, there is a read-write *workspace* of $O(s)$ words, where $s \in \{1, \dots, n\}$ is a given parameter. In a time-space trade-off, we are interested in how the running time of an algorithm improves as s varies from 1 to n.

We present a time-space trade-off for computing the *Euclidean minimum spanning tree* (EMST) of a set V of n sites in the plane. We present an algorithm that computes EMST(V) using $O(n^3 \log s/s^2)$ time and $O(s)$ words of workspace. Our algorithm uses the fact that EMST(V) is a subgraph of the bounded-degree *relative neighborhood graph* of V, and applies Kruskal's MST algorithm on it. To achieve this with limited workspace, we introduce a compact representation of planar graphs, called an *s-net* which allows us to manipulate its component structure during the execution of the algorithm.

Keywords: Euclidean minimum spanning tree
Relative neighborhood graph · Time-space trade-off
Limited workspace model · Kruskal's algorithm

1 Introduction

Given n sites in the plane, their *Euclidean minimum spanning tree* (EMST), is the minimum spanning tree with the sites as vertices, where the weight of the edge between two sites is their Euclidean distance. This problem is at the core of computational geometry and has been a classical problem taught in almost every first year lecture on the subject. Several classical algorithms are known that can compute EMST(V) in $O(n \log n)$ time using $O(n)$ words of workspace [11].

In this work, we revisit this problem, and design algorithms to compute the EMST in a memory-constrained model, where only few extra variables are allowed to be used during the execution of the algorithm. This kind of algorithms

B. Banyassady and W. Mulzer were supported in part by DFG project MU/3501/2.
L. Barba was supported by the ETH Postdoctoral Fellowship.

not only provides an interesting trade-off between running time and memory needed, but also is very useful in portable devices where important hardware constraints are present.

A significant amount of research was focused on the design of algorithms using few variables. Many of them dating from the 1970s, when memory used to be an expensive commodity. While in recent days the cost has substantially been reduced, the amount of data has increased, and the size of some devices has been dramatically reduced. Sensors and small devices where larger memories are neither possible nor desirable have proliferated in recent years. In addition, when working on inputs that do not fit in the local memory of our computer, it is often the case that data is simultaneously accessed by several devices. Moreover, even if a device is procured with a large memory, it might still be preferable to limit the number of write operations. Writing to flash memory is slow and costly, and may also reduce the lifetime of the memory. Additionally, if the input is stored on removable devices, write-access may not be allowed due to technical or security reasons. Therefore, while many memory-constrained models exist, the general scheme is the following: The input resides in a read-only memory where data cannot be modified by the algorithm. The algorithms are allowed to store a few variables that reside in a local memory and can be modified as needed to solve the problem (usually called *workspace*). Since the output may also not fit in our local memory, the model provides us with a write-only memory where the desired output is sequentially reported by the algorithm.

In general, one might consider algorithms that are allowed to use a workspace of $O(s)$ *words* for some parameter s, where a word is a collection of bits and is large enough to contain either an input item (such as a point coordinate) or a pointer into the input structure (of logarithmic size on the length of the input). The goal is then to design algorithms whose running time decreases as s increases, and that provide a nice trade-off between workspace size and running time.

Our results. For the case of EMST, Asano et al. [6] proposed an algorithm to compute the EMST of a set of n given sites in $O(n^3)$ time using a workspace of $O(1)$ words. In this paper, we revisit this problem and provide a time-space trade-off. Our algorithm computes the EMST in $O(n^3 \log s/s^2)$ time using $O(s)$ additional words of workspace. This algorithm provides a smooth transition between the $O(n^3)$ time algorithm [6] with constant words of workspace and the $O(n \log n)$ time algorithm [11] using a workspace of $O(n)$ words.

As the main tool to achieve this running time, we introduce a compact representation of planar graphs, called an *s-net*. The main idea is to carefully choose a "dense" set of s edges of the graph for which we remember their face incidences. That is, we store whether or not any of these edges are incident to the same face of the graph. Moreover, the density property of this s-net guarantees that no path can walk along a face of the graph for long without reaching an edge of the s-net. This allows us to "quickly" find the face of the graph that any given edge lies on. More specifically, we use this structure to speed up the implementation of Kruskal's EMST algorithm on planar graphs using limited workspace. Recall that in this algorithm, edges are added in increasing order to an auxiliary graph. Moreover, for each of them we need to find out whether or not its endpoints lie on the same component of this auxiliary graph when the edge is inserted. If the original graph is planar, then this amounts to testing whether or not these endpoints are incident to the same face of the graph—a task for which the compact representation of the s-net allows us to obtain time-space trade-offs to compute the EMST of planar graphs. While the s-net is designed to speed up Kruskal's algorithm, this structure is of independent interest as it provides a compact way to represent planar graphs that can be exploited by other algorithms.

Related work. The study of constant-workspace algorithm started with the introduction of the complexity class LOGSPACE [3]. After that, many classic problems were studied in this setting. Selection and sorting were among the first such problems [13,20–22]. In graph theory, Reingold [23] solved a long standing problem, and showed that connectivity in an undirected graph can be tested using constant workspace. The model was made popular in computational geometry by Asano et al. [6] who presented several algorithms to compute classic geometric data structures in the constant-workspace model. Algorithms with time-space trade-off for many of these problems were presented in subsequent years [1,2,4,5,7–10,15,16,18], with the notable exception of the problem of computing the EMST which is finally addressed in this paper.

2 Preliminaries and Definitions

Let V be a set of n points (sites) in the plane. The *Euclidean minimum spanning tree* of V, EMST(V), is the minimum spanning tree of the complete graph G on V, where the edges are weighted by the Euclidean distance between their endpoints. We assume that V is in general position, i.e., the edge lengths in G are pairwise distinct, thus EMST(V) is unique. Given V, we can compute EMST(V) in $O(n \log n)$ time using $O(n)$ words of workspace [11].

The *relative neighborhood graph* of V, RNG(V), is the undirected graph with vertex set V obtained by connecting two sites $u, v \in V$ with an edge if and only if there is no site $w \in V \backslash \{u, v\}$ such that both $|uw|$ and $|vw|$ is less than $|uv|$, where $|uv|$ denotes the Euclidean distance between u and v [24]. This is also known as the *empty lens* property, where the *lens* between u and v is the intersection of the disks of radius $|uv|$ centered at both u and v; see Fig. 1. One can show that a plane embedding of RNG(V) is obtained by drawing the edges as straight line segments between the corresponding sites in V. Furthermore, each vertex in RNG(V) has at most six neighbors, so that RNG(V) has $O(n)$ edges. We will denote the number of those edges by m. It is well-known that EMST(V) is a subgraph of RNG(V). In particular, this implies that RNG(V) is connected. Given V, we can compute RNG(V) in $O(n \log n)$ time using $O(n)$ words of workspace [17,19,24].

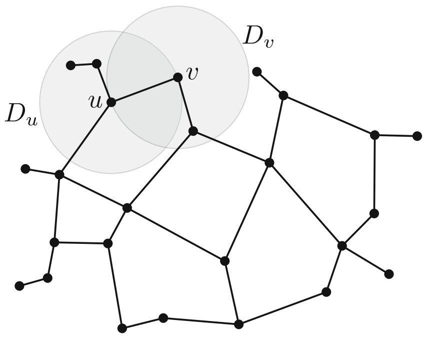


Fig. 1. The RNG for a set of sites V. The disks D_u and D_v have radius $|uv|$ and are centered at u and v, respectively. The edge uv is in RNG(V), since there is no site in V that lies in the lens $D_u \cap D_v$.

Recall the classic algorithm by Kruskal to find EMST(V) [14]: we start with an empty forest T, and we consider the edges of RNG(V) one by one, by increasing weight. In each step, we insert the current edge $e = vw$ into T if and only if there is no path between v and w in T. In the end, T will be EMST(V). Since EMST(V) is a subgraph of RNG(V), it suffices to consider only the edges of RNG(V). Thus, Kruskal's algorithm needs to consider $m = O(n)$ edges and runs in $O(n \log n)$ time, using $O(n)$ words of workspace.

Let $s \in \{1, \dots, n\}$ be a parameter, and assume that we are given a set V of n sites in general position (as defined above) in a read-only array. The goal is to find EMST(V), with $O(s)$ words of workspace. We use RNG(V) in order to compute EMST(V). By general position, the edge lengths in RNG(V) are pairwise distinct. Thus, we define $E_R = e_1, \dots, e_m$ to be the sorted sequence of the edges in RNG(V), in increasing order of length. For $i \in \{1, \dots, m\}$, we define RNG$_i$ to be the subgraph of RNG(V) with vertex set V and edge set $\{e_1, \dots, e_{i-1}\}$.

In the limited workspace model, we cannot store RNG$_i$ explicitly. Instead, we resort to the *computing instead of storing* paradigm [6]. That is, we completely compute the next batch of edges in E_R whenever we need new edges of RNG(V) in Kruskal's algorithm. To check whether a new edge $e_i \in E_R$ belongs to EMST(V), we need to check if e_i connects two distinct components of RNG$_i$. To do this with $O(s)$ words of workspace, we will use a succinct representation of its component structure; see below. In our algorithm, we represent each edge $e_i \in E_R$ by two directed *half-edges*. The two half-edges are oriented in opposite directions such that the face incident a half-edge lies to the left of it. We call the endpoints of a half-edge the *head* and the *tail* such that the half-edge is directed from the tail endpoint to the head endpoint. Obviously, each half-edge in RNG$_i$ has an opposing partner. However, in our succinct representation, we will rely on individual half-edges. Throughout the paper, directed half-edges will be denoted as $\overrightarrow{e}$, and undirected edges as e. For a half-edge $\overrightarrow{e} = \overrightarrow{uv}$ with $u, v \in V$, we call v the *head* of $\overrightarrow{e}$, and u the *tail* of $\overrightarrow{e}$.

3 The Algorithm

Before we discuss our algorithm, we explain how to compute batches of edges in $\mathrm{RNG}(V)$ using $O(s)$ words of workspace. A similar technique has been used previously in the context of Voronoi diagrams [8].

Lemma 3.1. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. Given a set $Q \subseteq V$ of s sites, we can compute for each $u \in Q$ the at most six neighbors of u in $\mathrm{RNG}(V)$ in total time $O(n \log s)$, using $O(s)$ words of workspace.*

Proof. The algorithm uses $\lceil n/s \rceil$ *steps*. In each step, we process a *batch* of s sites of $V = V_1 \cup \ldots \cup V_{\lceil n/s \rceil}$, and produce at most six candidates for each site of Q to be in $\mathrm{RNG}(V)$. In the first step, we take the first batch $V_1 \subseteq V$ of s sites, and we compute $\mathrm{RNG}(Q \cup V_1)$. Because both Q and V_1 have at most s sites, we can do this in $O(s \log s)$ time using $O(s)$ words of workspace using standard algorithms. For each $u \in Q$, we remember the at most six neighbors of u in $\mathrm{RNG}(Q \cup V_1)$. Notice that for each pair $u \in Q, v \in V_1$, if the edge uv is not in $\mathrm{RNG}(Q \cup V_1)$, then the lens of u and v is non-empty. That is, there is a witness among the points of $Q \cup V_1$ that certifies that uv is not an edge of $\mathrm{RNG}(V)$. Let N_1 be the set containing all neighbors in $\mathrm{RNG}(Q \cup V_1)$ of all sites in Q. Storing N_1, the set of candidate neighbors requires $O(s)$ words of workspace.

Then, in each step $j = 2, \ldots, O(n/s)$, we take next batch $V_j \subseteq V$ of s sites, and compute $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ in $O(s \log s)$ time using $O(s)$ words of space. For each $u \in Q$, we store the set of at most six neighbors in this computed graph. Additionally, we let N_j be the set containing all neighbors in $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ of all sites in Q. Note that N_j, the set of candidate neighbors, consists of $O(s)$ sites as each site in Q has degree at most six in the computed graph.

Therefore, after $\lceil n/s \rceil$ steps, we are left with at most six candidate neighbors for each site in Q. As mentioned above, for a pair $u \in Q, v \in V$, if v is not among the candidate neighbors of u, then at some point in the construction there was a site witnessing that the lens of u and v is non-empty. Therefore, only the sites which are in the set of candidate neighbors can define edges of $\mathrm{RNG}(V)$. However, all the candidate neighbors are not necessarily the neighbors in $\mathrm{RNG}(V)$ of sites in Q.

To obtain the edges of $\mathrm{RNG}(V)$ incident to the sites of Q, we take each site in Q and its corresponding neighbors in $N_{\lceil n/s \rceil}$. Then, we go again through the entire set $V = V_1 \cup \ldots \cup V_{\lceil n/s \rceil}$ in batches of size s: for each $u \in Q$, we test the at most six candidate neighbors in $N_{\lceil n/s \rceil}$ against all elements of the current batch to test the empty-lens property. After going through all sites, the candidates that maintained the empty-lens property throughout define the edges of $\mathrm{RNG}(V)$ incident to the sites of Q. Since we use $O(s \log s)$ time per step, and since there are $\lceil n/s \rceil$ steps, the total running time is $O(n \log s)$ using $O(s)$ words of workspace. □

Through repeated application of Lemma 3.1, we can enumerate the edges of $\mathrm{RNG}(V)$ by increasing lengths.

Lemma 3.2. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. Let $E_R = e_1, e_2, \dots, e_m$ be the sequence of edges in* $\mathrm{RNG}(V)$*, by increasing length. Let $i \geq 1$. Given e_{i-1} (or $\perp$, if $i = 1$), we can find the edges $e_i, \dots, e_{i+s-1}$ in $O(n^2 \log s/s)$ time using $O(s)$ words of workspace.*[1]

Proof. By applying Lemma 3.1 $O(n/s)$ times, we can generate all the edges of $\mathrm{RNG}(V)$. Because we obtain the edges in batches of size $O(s)$, each taking $O(n \log s)$ time, the total time to compute all the edges amounts to $O(n^2 \log s/s)$. During this process, we find the edges $e_i, \dots, e_{i+s-1}$ of E_R. This can be done with a trick by Chan and Chen [12], similar to the procedure in the second algorithm in [7]. More precisely, whenever we produce new edges of $\mathrm{RNG}(V)$, we store the edges that are longer than e_{i-1} in an array A of size $O(s)$. Whenever A contains more than $2s$ elements, we use a linear time selection procedure to remove all edges of rank larger than s [14]. This needs $O(s)$ operations per step. We repeat this procedure for $O(n/s)$ steps, giving total time $O(n)$ for selecting the edges. In the end, we have $e_i, \dots, e_{i+s-1}$ in A, albeit not in sorted order. Thus, we sort the final A in $O(s \log s)$ time. The running time is dominated by the time needed to compute the edges of $\mathrm{RNG}(V)$, so the claim follows. $\square$

Lemma 3.2, together with the techniques from the original constant workspace EMST-algorithm by Asano et al. [6], already leads to a simple time-space trade-off for computing $\mathrm{EMST}(V)$. Recall that we represent the edges of $\mathrm{RNG}(V)$ as pairs of opposing half-edges, such that the face incident to a half-edge lies to its left. For $i \in \{1, \dots, m\}$, a *face-cycle* in RNG_i is the circular sequence of half-edges that bounds a face in RNG_i. All half-edges in a face-cycle are oriented in the same direction, and RNG_i can be represented as a collection of face-cycles; see Fig. 2. Asano et al. [6] observe that to run Kruskal's algorithm on $\mathrm{RNG}(V)$, it suffices to know the structure of the face-cycles.

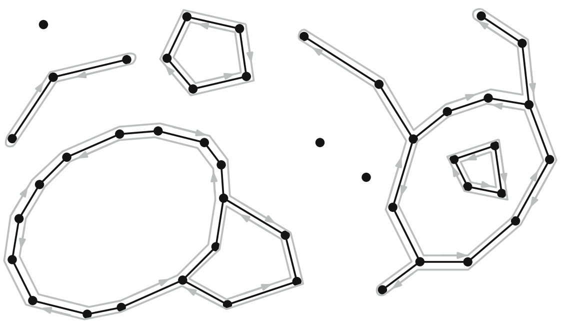


Fig. 2. A schematic drawing of RNG_i is shown in black. The face-cycles of this graph are shown in gray. All the half-edges of a face-cycle are directed according to the arrows.

[1] Naturally, if $i + s - 1 > m$, we report the edges $e_i, \dots, e_m$.

Observation 3.3. *Let $i \in \{1, \dots, m\}$. The edge $e_i \in E_R$ belongs to* $\mathrm{EMST}(V)$ *if and only if there is no face-cycle C in* RNG_i *such that both endpoints of e_i lie on C.*

Proof. Let u and v be the endpoints of e_i. If there is a face-cycle C in RNG_i that contains both u and v, then e_i clearly does not belong to $\mathrm{EMST}(V)$. Conversely, suppose there is no face-cycle in RNG_i containing both u and v. Thus, any two face-cycles C_u and C_v such that u lies on C_u and v lies on C_v must be distinct. Since $\mathrm{RNG}(V)$ is plane, C_u and C_v must belong to two different connected components of RNG_i, and e_i is an edge of $\mathrm{EMST}(V)$. □

Observation 3.3 tells us that we can identify the edges of $\mathrm{EMST}(V)$ if we can determine, for each $i \in \{1, \dots, m\}$, the face-cycles of RNG_i that contain the endpoints of e_i. To accomplish this task, we use the next lemma to traverse the face-cycles.

Lemma 3.4. *Let $i \in \{1, \dots, m\}$. Suppose we are given $e_i \in E_R$ and a half-edge $\overrightarrow{f} \in \mathrm{RNG}_i$, as well as the at most six edges incident to the head of $\overrightarrow{f}$ in* $\mathrm{RNG}(V)$. *Let C be the face-cycle of* RNG_i *that $\overrightarrow{f}$ lies on. We can find the half-edge $\overrightarrow{f'}$ that comes after $\overrightarrow{f}$ on C, in $O(1)$ time using $O(1)$ words of workspace.*

Proof. Let w be the head of $\overrightarrow{f}$. By comparing the edges incident to w with e_i, we identify the incident half-edges of w in RNG_i, in $O(1)$ time. Then, among them we pick the half-edge $\overrightarrow{f'}$ which has the smallest clockwise angle with $\overrightarrow{f}$ around w and has w as its tail. This takes $O(1)$ time using $O(1)$ words of workspace. □

For $j \geq i \geq 1$, we define *predecessor* and *successor* of e_j in RNG_i regarding each endpoint w of e_j as follows: the predecessor $\overrightarrow{p_w}$ of e_j is the half-edge in RNG_i which has w as its head and is the first half-edge encountered in a counterclockwise sweep from e_j around w. The successor $\overrightarrow{s_w}$ of e_j is the half-edge in RNG_i which has w as its tail and is the first half-edge encountered in a clockwise sweep from e_j around w; see Fig. 3. If there is no edge incident to w in RNG_i, we set $p_w, s_w = \perp$.

From our observations so far, we can already derive a simple time-space trade-off for computing $\mathrm{EMST}(V)$.

Theorem 3.5. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \dots, n\}$ be a parameter. We can output all the edges of* $\mathrm{EMST}(V)$*, in sorted order, in $O(n^3 \log s/s)$ time using $O(s)$ words of workspace.*

Proof. We simulate Kruskal's algorithm on $\mathrm{RNG}(V)$. For this, we take batches of s edges, sorted by increasing length, and we report the edges of $\mathrm{EMST}(V)$ in each batch. Let $E_R = e_1, \dots, e_m$ be the edges of $\mathrm{RNG}(V)$, sorted by length. To determine whether an edge $e_i \in E_R$ is in $\mathrm{EMST}(V)$, we apply Observation 3.3, i.e., we determine whether the endpoints of e_i are on two distinct face-cycles of the corresponding RNG_i. To do this, we process E_R in batches of s edges, and for each edge, we perform a walk along the face-cycle that contains one endpoint

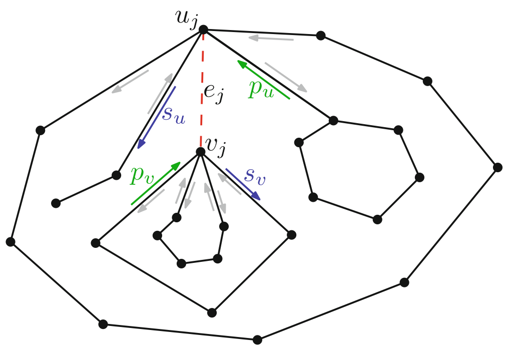


Fig. 3. A schematic drawing of RNG_i is shown in black. The endpoint $w = u, v$ of e_j identifies the half-edges p_w and s_w as the predecessor and the successor of e_j. They are shown in green and blue, respectively. (Color figure online)

of e_i until we either encounter the other endpoint of e_i or until we are back at the starting point of our walk.

More precisely, we proceed as follows: first, we use Lemma 3.2 to find the next batch $e_i, \ldots, e_{i+s-1}$ of s edges in E_R, in $O(n^2 \log s/s)$ time. For each such edge e_j, we pick an endpoint $u_j \in V$. Using Lemma 3.1, we find for each u_j first the incident edges in $\mathrm{RNG}(V)$, and then the incident edges in RNG_j (by comparing the edges from $\mathrm{RNG}(V)$ with e_j). Then, we identify the successor of each e_j in RNG_j (if it exists), and we perform s parallel walks, where walk j takes place in RNG_j. In each step, we have s current half-edges, and we use Lemmas 3.1 and 3.4 to advance each half-edge along its face-cycle. This takes $O(n \log s)$ operations. A walk j continues until we either encounter the other endpoint of e_j or until we arrive at the predecessor of e_j in RNG_j. In the latter case, e_j is in $\mathrm{EMST}(V)$, and we report it. In the former case, e_j is not in $\mathrm{EMST}(V)$. Since there are $O(n)$ half-edges in $\mathrm{RNG}(V)$, it takes $O(n)$ steps to conclude all the walks. If follows that we can process a single batch of edges in $O(n^2 \log s)$ time. We have $O(n/s)$ many batches, so the total running time of the algorithm is $O(n^3 \log s/s)$, using $O(s)$ words of workspace. □

Theorem 3.5 is clearly not optimal: for the case of linear space $s = n$, we get a running time of $O(n^2 \log n)$, although we know that it should take $O(n \log n)$ time to find $\mathrm{EMST}(V)$. Can we do better? The bottleneck in Theorem 3.5 is the time needed to perform the walks in the partial relative neighborhood graphs RNG_j. In particular, such a walk might take up to $\Omega(n)$ steps, leading to a running time of $\Omega(n^2 \log s)$ for processing a single batch. To avoid this, we will maintain a compressed representation of the partial relative neighborhood graphs that allow us to reduce the number of steps in each walk to $O(n/s)$.

Let $i \in \{1, \ldots, m\}$. An *s-net* N for RNG_i is a collection of half-edges, called *net-edges*, in RNG_i that has the following two properties: (i) each face-cycle in RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contains at least one net-edge; and (ii) for any net-edge $\overrightarrow{e} \in N$, let C be the face-cycle of RNG_i with $\overrightarrow{e}$. Then, between the head of $\overrightarrow{e}$ and the tail of the next net-edge on C, there are at least $\lfloor n/s \rfloor$ and at most $2\lfloor n/s \rfloor$ other half-edges on C. Note that the next net-edge on C after $\overrightarrow{e}$ could be possibly $\overrightarrow{e}$ itself. In particular, this implies that face-cycles with less than $\lfloor n/s \rfloor$ edges contain no net-edge. The following observation records two important properties of s-nets.

Observation 3.6. *Let $i \in \{1, \ldots, m\}$, and let N be an s-net for* RNG_i. *Then, (N1) N has $O(s)$ half-edges; and (N2) let $\overrightarrow{f}$ be a half-edge of* RNG_i, *and let C be the face-cycle that contains it. Then, it takes at most $2\lfloor n/s \rfloor$ steps along C from the head of $\overrightarrow{f}$ until we either reach a net-edge or the tail of $\overrightarrow{f}$.*

Proof. Property (ii) implies that only face-cycle of RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contain net-edges. Furthermore, on these face-cycles, we can uniquely charge $\Theta(n/s)$ half-edges to each net-edge, again by (ii). Thus, since there are $O(n)$ half-edges in total, we have the first statement $|N| = O(s)$.

For the second statement, we first note that if C contains less than $2\lfloor n/s \rfloor$ half-edges, the claim holds trivially. Otherwise, C contains at least one net-edge, by property (i). Now, property (ii) shows that we reach a net-edge in at most $2\lfloor n/s \rfloor$ steps from $\overrightarrow{f}$. □

By Observation 3.6, we can store an s-net in $O(s)$ words of workspace. This makes the concept of s-net useful in our time-space trade-off. Now, we can use the s-net in order to speed up the processing of a single batch. The next lemma shows how this is done:

Lemma 3.7. *Let $i \in \{1, \ldots, m\}$, and let $E_{i,s} = e_i, \ldots, e_{i+s-1}$ be a batch of s edges from E_R. Suppose we have an s-net N for* RNG_i *in our workspace. Then, we can determine which edges from $E_{i,s}$ belong to* $\mathrm{EMST}(V)$, *using $O(n^2 \log s/s)$ time and $O(s)$ words of workspace.*

Proof. Let F be the set of half-edges that contains all net-edges from N, as well as, for each *batch-edge* $e_j \in E_{i,s}$, the two successors of e_j in RNG_i, one for each endpoint of e_j. By definition, we have $|F| = O(s)$, and it takes $O(n \log s)$ time to compute F, using Lemma 3.1. Now, we perform parallel walks through the face-cycles of RNG_i, using Lemmas 3.1 and 3.4. We have one walk for each half-edge in F, and each walk proceeds until it encounters the tail of a half-edge from F (including the starting half-edge itself). By Lemma 3.4, in each step of these parallel walks we need $O(n \log s)$ time to find the next edge on the face-cycle and then we need $O(s \log s)$ time to check whether these new edges are in F. Because F contains the net-edges of N, by property (N2), each walk finishes after $O(n/s)$ steps, and thus the total time for this procedure is $O(n^2 \log s/s)$.

Next, we build an auxiliary *undirected* graph H, as follows: the vertices of H are the endpoints of the half-edges in F. Furthermore, H contains undirected edges for all the half-edges in F and additional *compressed edges*, that represent the outcomes of the walks: if a walk started from the head u of a half-edge in F and ended at the tail v of a half-edge in F, we add an edge from u to v in H, and we label it with the number of steps that were needed for the walk. Thus, H

contains F*-edges*, and *compressed edges*; see Fig. 4. Clearly, after all the walks have been performed, we can construct H in $O(s)$ time, using $O(s)$ words of workspace.

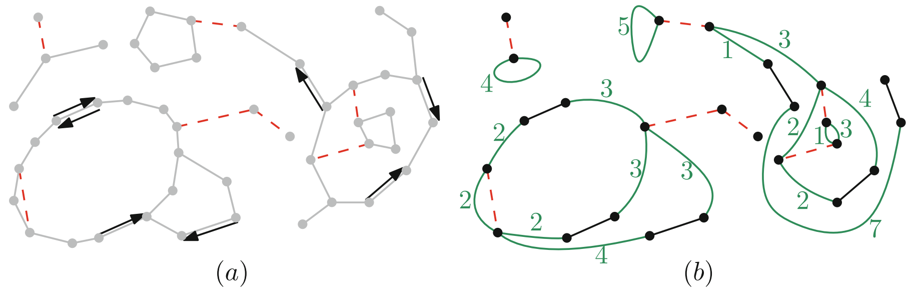


Fig. 4. (a) A schematic drawing of RNG_i is shown in gray. The half-edges of N are in black and the edges of the next batch $E_{i,s}$ are dashed red segments. (b) The auxiliary graph H including the batch-edges (in red). The graph H contains the net-edges (in black), and the successors of batch-edges and the compressed edges (which are combined in green paths in this picture). (Color figure online)

Next, we use Kruskal's algorithm to insert the batch-edges of $E_{i,s}$ into H. This is done as follows: we determine the connected components of H, in $O(s)$ time using depth-first search. Then, we insert the batch-edges into H, one after another, in sorted order. As we do this, we keep track of how the connected components of H change, using a union-find data structure [14]. Whenever a new batch-edge connects two different connected components, we output it as an edge of $\mathrm{EMST}(V)$. Otherwise, we do nothing. Note that even though H may have a lot more components than RNG_i, the algorithm is still correct, by Observation 3.3. This execution of Kruskal's algorithm, and updating the structure of connected components of H takes $O(s \log s)$ time, which is dominated by the running time of $O(n^2 \log s/s)$ from the first phase of the algorithm. □

Finally, we need to explain how to maintain the s-net during the algorithm. The following lemma shows how we can compute an s-net for RNG_{i+s}, provided that we have an s-net for RNG_i and the graph H described in the proof of Lemma 3.7, for each $i \in \{1, \ldots, m\}$.

Lemma 3.8. *Let $i \in \{1, \ldots, m\}$, and suppose we have the graph H derived from* RNG_i *as above, such that all batch-edges have been inserted into H. Then, we can compute an s-net N for* RNG_{i+s} *in time $O(n^2 \log s/s)$, using $O(s)$ words of workspace.*

Proof. By construction, all *big* face-cycles of RNG_{i+s}, which are the faces with at least $\lfloor n/s \rfloor + 1$ half-edges appear as faces in H. Thus, by walking along all faces in H, and taking into account the labels of the compressed edges, we can determine these big face-cycles in $O(s)$ time. The big face-cycles are represented through sequences of F-edges, compressed edges, and batch-edges. For each such sequence, we determine the positions of the half-edges for the new s-net N, by spreading the half-edges equally at distance $\lfloor n/s \rfloor$ along the sequence, again taking the labels of the compressed edges into account. Since the compressed edges have length $O(n/s)$, for each of them, we create at most $O(1)$ new net-edges. Now that we have determined the positions of the new net-edges on the face-cycles of RNG_{i+s}, we perform $O(s)$ parallel walks in RNG_{i+s} to actually find them. Using Lemma 3.4, this takes $O(n^2 \log s/s)$ time. □

We now have all the ingredients for our main result which provides a smooth trade-off between the cubic time algorithm in constant workspace and the classical $O(n \log n)$ time algorithm with $O(n)$ words of workspaces.

Theorem 3.9. *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. We can output all the edges of* $\mathrm{EMST}(V)$*, in sorted order, in $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace.*

Proof. This follows immediately from Lemmas 3.7 and 3.8, because we need to process $O(n/s)$ batches of edges from E_R. □

For our algorithm, it suffices to update the s-net every time that a new batch is considered. It is however possible to maintain the s-net and the auxiliary graph H through insertions of single edges. This allows us to handle graphs constructed incrementally and maintain their compact representation using $O(s)$ workspace words. We believe this is of independent interest and can be used by other algorithms for planar graphs in the limited-workspace model.

Acknowledgments. This work was initiated at the Fields Workshop on Discrete and Computational Geometry, held July 31–August 04, 2017, at Carleton university. The authors would like to thank them and all the participants of the workshop for inspiring discussions and for providing a great research atmosphere.

References

1. Ahn, H.-K., Baraldo, N., Oh, E., Silvestri, F.: A time-space trade-off for triangulations of points in the plane. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 3–12. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_1
2. Aronov, B., Korman, M., Pratt, S., van Renssen, A., Roeloffzen, M.: Time-space trade-offs for triangulating a simple polygon. In: Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), pp. 30:1–30:12 (2016)
3. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)
4. Asano, T., Buchin, K., Buchin, M., Korman, M., Mulzer, W., Rote, G., Schulz, A.: Memory-constrained algorithms for simple polygons. Comput. Geom. **46**(8), 959–969 (2013)

5. Asano, T., Kirkpatrick, D.: Time-space tradeoffs for all-nearest-larger-neighbors problems. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 61–72. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_6
6. Asano, T., Mulzer, W., Rote, G., Wang, Y.: Constant-work-space algorithms for geometric problems. J. Comput. Geom. **2**(1), 46–68 (2011)
7. Bahoo, Y., Banyassady, B., Bose, P., Durocher, S., Mulzer, W.: Time-space trade-off for finding the k-visibility region of a point in a polygon. In: Poon, S.-H., Rahman, M.S., Yen, H.-C. (eds.) WALCOM 2017. LNCS, vol. 10167, pp. 308–319. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53925-6_24
8. Banyassady, B., Korman, M., Mulzer, W., van Renssen, A., Roeloffzen, M., Seiferth, P., Stein, Y.: Improved time-space trade-offs for computing Voronoi diagrams. In: Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS), pp. 9:1–9:14 (2017)
9. Barba, L., Korman, M., Langerman, S., Sadakane, K., Silveira, R.I.: Space-time trade-offs for stack-based algorithms. Algorithmica **72**(4), 1097–1129 (2015)
10. Barba, L., Korman, M., Langerman, S., Silveira, R.I.: Computing a visibility polygon using few variables. Comput. Geom. **47**(9), 918–926 (2014)
11. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.H.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
12. Chan, T.M., Chen, E.Y.: Multi-pass geometric algorithms. Discrete Comput. Geom. **37**(1), 79–102 (2007)
13. Chan, T.M., Munro, J.I., Raman, V.: Selection and sorting in the “restore” model. In: Proceedings of the 25th Annual ACM-SIAM symposium Discrete Algorithms (SODA), pp. 995–1004 (2014)
14. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
15. Darwish, O., Elmasry, A.: Optimal time-space tradeoff for the 2D convex-hull problem. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 284–295. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_24
16. Har-Peled, S.: Shortest path in a polygon using sublinear space. J. Comput. Geom. **7**(2), 19–45 (2016)
17. Jaromczyk, J.W., Toussaint, G.T.: Relative neighborhood graphs and their relatives. Proc. IEEE **80**, 1502–1517 (1992)
18. Korman, M., Mulzer, W., van Renssen, A., Roeloffzen, M., Seiferth, P., Stein, Y.: Time-space trade-offs for triangulations and Voronoi diagrams. Comput. Geom. (2017, to appear)
19. Mitchell, J.S.B., Mulzer, W.: Proximity algorithms. In: Goodman, J.E., O’Rourke, J., Tóth, C.D. (eds.) Handbook of Discrete and Computational Geometry, 3rd edn, pp. 849–874. CRC Press, Boca Raton (2017)
20. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. Theoret. Comput. Sci. **12**(3), 315–323 (1980)
21. Munro, J.I., Raman, V.: Selection from read-only memory and sorting with minimum data movement. Theoret. Comput. Sci. **165**(2), 311–323 (1996)
22. Pagter, J., Rauhe, T.: Optimal time-space trade-offs for sorting. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 264–268 (1998)
23. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 17 (2008)
24. Toussaint, G.T.: The relative neighbourhood graph of a finite planar set. Pattern Recogn. **12**(4), 261–268 (1980)

Approximate Nearest Neighbor Search for ℓ_p-Spaces $(2 < p < \infty)$ via Embeddings

Yair Bartal[1] and Lee-Ad Gottlieb[2](✉)

[1] Hebrew University, Jerusalem, Israel
yair@cs.huji.ac.il
[2] Ariel University, Ariel, Israel
leead@ariel.ac.il

Abstract. While the problem of approximate nearest neighbor search has been well-studied for Euclidean space and ℓ_1, few non-trivial algorithms are known for ℓ_p when $2 < p < \infty$. In this paper, we revisit this fundamental problem and present approximate nearest-neighbor search algorithms which give the best known approximation factor guarantees in this setting.

1 Introduction

Nearest neighbor search (NNS) is one of the basic operations computed on data sets comprising numeric vectors, i.e. points. The problem asks to preprocess a d-dimensional set V of $n = |V|$ vectors residing in a certain space M, so that given a new query point $q \in M$, a point nearest to q in V can be located efficiently. This problem has applications in data mining, database queries and related fields.

When the ambient space M is a high-dimensional ℓ_p-space,[1] NNS may require significant computation time, and this is due to the inherent complexity of the metric. For example, for Euclidean vectors (ℓ_2-space), Clarkson [12] gave an $O(n^{\lceil d/2 \rceil (1+\delta)})$ size data structure that answers exact NNS queries in $O(\log n)$ time (with constant factors in the query time depending on constant $\delta > 0$), and claimed that the exponential dependence on d is a manifestation of Bellman's [7] "curse of dimensionality." This has led researchers to consider the c-approximate nearest neighbor problem (ANN), where the goal is to find a point in V whose distance to q is within a factor c of the distance to q's true nearest neighbor in V. In the Euclidean setting, celebrated results of Kushilevitz, Ostrovsky and Rabani [27] (see also [31,32]) and Indyk and Motwani [18,22] achieved polynomial-size data structures which return a $(1+\varepsilon)$-ANN in query time polynomial in $d \log n$ (when $\varepsilon > 0$ is any constant). These results can be extended to all ℓ_p with $1 \le p \le 2$.

Y. Bartal is supported in part by an Israel Science Foundation grant #1817/17. L.-A. Gottlieb is supported in part by an Israel Science Foundation grant #755/15.

[1] This is a space equipped with a Minkowski norm, which defines the distance between two d-dimensional vectors x, y as $\|x - y\|_p = (\sum_{i=1}^{d} |x_i - y_i|^p)^{1/p}$.

However, the more difficult regime of $p > 2$ is significantly less well understood. Recalling that for any vector v and $p > 2$, $d^{\frac{1}{p}-\frac{1}{2}}\|v\|_2 \le \|v\|_p \le \|v\|_2$, we conclude that simply running an ℓ_2 ANN algorithm on $V \subset \ell_p$ (that is, treating V as if it resided in ℓ_2) will return an $O(d^{\frac{1}{2}-\frac{1}{p}})$-ANN in time polynomial in $d \log n$. If we allow exponential space, then a $(1+\varepsilon)$-ANN can be found in query time $O(d \log n)$ and space $\varepsilon^{-O(d)}n$ by utilizing an approximate Voronoi diagram (AVD) [4,18] (These structures were developed for Euclidean spaces, but apply to all ℓ_p, $p \ge 1$ as well [20].) For ℓ_∞, Indyk [21] gave a polynomial-size structure which answers $O(\log \log d)$-approximate queries in $d \log^{O(1)} n$ query time, and remarkably there are indications that this bound may be optimal [2]. Since for any vector v we have $\|v\|_\infty \le \|v\|_p \le d^{1/p}\|v\|_\infty$, Indyk's ℓ_∞ structure gives a $O(d^{1/p} \log \log d)$-ANN algorithm for all $p > 2$, and in particular an $O(\log \log d)$-ANN whenever $p = \Omega(\log d)$. Finally, Andoni [3] gave an embedding-based algorithm for $p > 2$ which combined with Indyk's result for l_∞ returns an $O(\log \log d \lceil \log_t^{1/p} n \rceil)$-ANN in time $O(td \log^2 n)$ for any $t > 2$.

Our contributions. We revisit the problem of ANN in ℓ_p spaces for $2 < p < \infty$, and give improvements over what was previously known. Note that the ℓ_p-norm for this regime finds application in fields such as image resolution [16,17], time-series comparison [33], and k-means clustering [14]. We are interested in polynomial-size structures that have query time polynomial in $d \log n$ (with both space and query time independent of p). Hence we shall make the simplifying assumptions[2] that $d = \omega(\log n)$ and $d = n^{o(1)}$.

In Sect. 2.1, we give a simple presentation of Andoni's algorithm for ℓ_p spaces, showing how the max-stability property of Fréchet random variables can be exploited to give a randomized embedding from ℓ_p into ℓ_∞ that is non-contractive and has small expansion. By running Indyk's ℓ_∞ algorithm on the embedded space, Andoni obtains the approximation and runtime stated above; taking $t = d^{O(1)}$ gives a $O(\log \log d \lceil \log_d^{1/p} n \rceil)$-ANN in time polynomial in $d \log n$. Our first contribution (Sect. 2.2) is to improve on Andoni's result for subspaces of low intrinsic dimension, where we remove the dependence on $\log n$ in the approximation and replace it with a similar dependence on the doubling dimension of the subspace. We show that a $O\left(\log \log d \cdot \left\lceil \frac{\text{ddim} \log \text{ddim}}{\log d} \right\rceil^{1/p}\right)$-ANN can be found in polynomial time.

Having improved on Andoni's approach, which is based on Indyk's ℓ_∞ algorithm, we proceed to introduce an embedding which greatly extends the range of p for which the ℓ_2 algorithms are applicable. In Sect. 3, we introduce the Mazur map as an algorithmic tool. This mapping allows us to embed ℓ_p into ℓ_2, and we

[2] If $d = O(\log n)$ then AVDs may be used, and if $d = n^{\Omega(1)}$ then comparing the query point q to each point in V in a brute-force manner can be done in $O(dn) = d^{O(1)}$ time. (We recall also that there exists an oblivious mapping for all ℓ_p that embeds ℓ_p^m into ℓ_p^d for $d = \binom{n}{2}$ dimensions [5,15].) We also assume that $d = 2^{o(\text{ddim})}$, as otherwise a constant-factor approximation can be computed in polynomial time [13,19].

then solve ANN in the embedded space. Although the Mazur map induces distortion dependent on the diameter of the set, thereby confounding the ANN search, we show that the mapping can be applied to small low-diameter subproblems. Our final result is a polynomial-size structure which answers $2^{O(p)}$-approximate queries in time polynomial in $d \log n$ (Theorem 5). This yields non-trivial results for small p. Comparing this result with Andoni's algorithm:

– When $p = O(\log\log\log d + \sqrt{\log\log_d n})$, the $2^{O(p)}$-ANN algorithm is best.
– When $p = \Omega(\log\log\log d + \sqrt{\log\log_d n})$, the $O(\log\log d \log_d^{1/p} n)$-ANN algorithm is best.

Note that the worst case is when $p = \Theta(\log\log\log d + \sqrt{\log\log_d n})$, where the approximation ratio is $2^{O(p)} = \log\log d \cdot 2^{O(\sqrt{\log\log_d n})}$. This provides the first sub-logarithmic approximation bound for all values of p. When our doubling dimension bounds are taken into account, the worst case is achieved at $p = O(\log\log\log d + \sqrt{\log(\mathrm{ddim}/\log d)})$, where the approximation ratio is $\log\log d \cdot 2^{O(\sqrt{\log(\mathrm{ddim}/\log d)})}$.

1.1 Related Work

For Euclidean space, Chan [11] gave a deterministic construction which gives an $O(d^{3/2})$-ANN, in time $O(d^2 \log n)$ and using polynomial space (see also [8]). For ℓ_p $(p \geq 1)$, Neylon [30] gave an $O(d)$-ANN structure which runs in $O(d^2 \log n)$ time and uses $\tilde{O}(dn)$ space.

For ANN in general metric spaces, Krauthgamer and Lee [26] showed that the doubling dimension can be used to control the search runtime: For a metric point set S, they constructed a polynomial-size structure which finds an $O(1)$-ANN in time $2^{O(\mathrm{ddim}(S))} \log \Delta$, where $\Delta = \Delta(S)$ is the *aspect ratio* of S, the ratio between the maximum and minimum inter-point distances in S. The space requirements of this data structure were later improved by Beygelzimer *et al.* [9]. Har-Peled and Mendel [19] and Cole and Gottlieb [13] showed how to replace the dependence on $\log \Delta$ with dependence on $\log n$.

Subsequent to the public dissemination of our Mazur map algorithm, Naor and Rabani informed us that they independently discovered a NNS algorithm for $p > 2$, also using the Mazur map [29] (mentioned in [28, Remark 4.2]). We defer a detailed comparison until their algorithm is publicized. In personal communication, Assaf Naor broached the question of better dependence on p in the $2^{O(p)}$ approximation bound of Theorem 5. He noted that all *uniform* embeddings of ℓ_p $(p > 2)$ into ℓ_2 (such as the Mazur map) possess distortion exponential in p [28, Lemma 5.2], although non-uniform embeddings of ℓ_p into ℓ_2 may possess better distortion bounds. We have somewhat mitigated this problem by combining our algorithm with that of Andoni, which is an embedding into ℓ_∞.

1.2 Preliminaries

Embeddings and metric transforms. A much celebrated result for dimension reduction is the well-known l_2 flattening lemma of Johnson and Lindenstrauss

[24]: For every n-point subset of l_2 and every $0 < \varepsilon < 1$, there exists a mapping into l_2^k that preserves all inter-point distances in the set within factor $1+\varepsilon$, with target dimension $k = O(\varepsilon^{-2} \log n)$.

Following Batu *et al.* [6], we define an *oblivious* embedding to be an embedding which can be computed for any point of a database or query set, without knowledge of any other point in these sets. (This differs slightly from the definition put forth by Indyk and Naor [23].) Familiar oblivious embeddings include standard implementations of the Johnson-Lindenstrauss transform for l_2 [24], the dimension reduction mapping of Ostrovsky and Rabani [32] for the Hamming cube, and the embedding of Johnson and Schechtman [25] for ℓ_p, $p \le 2$.

An embedding of X into Y with *distortion* D is a mapping $f : X \to Y$ such that for all $x, y \in X$, $1 \le c \cdot \frac{d_Y(f(x),f(y))}{d_X(x,y)} \le D$, where c is any scaling constant. An embedding is *non-contractive* if this holds with $c \le 1$, and *non-expansive* (or *Lipschitz*) if $c \ge D$.

Doubling dimension. For a metric M, let $\lambda > 0$ be the smallest value such that every ball in M can be covered by λ balls of half the radius. λ is the *doubling constant* of M, and the *doubling dimension* of M is $\text{ddim}(M) = \log_2 \lambda$. Then clearly $\text{ddim}(M) = O(\log n)$. Note that while a low ℓ_p vector dimension implies a low doubling dimension – simple volume arguments demonstrate that ℓ_p metrics ($p \ge 1$) of dimension d have doubling dimension $O(d)$ – low doubling dimension is strictly more general than low ℓ_p dimension. We will often use the notation ddim when the ambient space is clear from context. The following packing property can be shown (see for example [26]):

Lemma 1. *Suppose that $S \subset M$ has minimum inter-point distance α, and let* $\text{diam}(S)$ *be the diameter of S. Then* $|S| \le \left(\frac{2\,\text{diam}(S)}{\alpha}\right)^{\text{ddim}(M)}$.

Nets and hierarchies. Given a point set S residing in metric space M, $S' \subset S$ is a γ-net of S if the minimum inter-point distance in S' is at least γ, while the distance from every point of S to its nearest neighbor in S' is less than γ. Let S have minimum inter-point distance 1. A *hierarchy* is a series of $\lceil \log \Delta \rceil$ nets (Δ being the aspect ratio of S), where each net S_i is a 2^i-net of the previous net S_{i-1}. The first (or bottom) net is $S_0 = S$, and the last (or top) net S_t contains a single point called the *root*. For two points $u \in S_i$ and $v \in S_{i-1}$, if $d(u,v) < 2^i$ then we say that u *covers* v, and this definition allows v to have multiple covering points in 2^i. The closest covering point of v is its *parent*. The distance from a point in S_i to its ancestor in S_j ($j > i$) is at most $\sum_{k=i+1}^{j} 2^k = 2 \cdot (2^j - 2^{i+1}) < 2 \cdot 2^j$.

Given S, a hierarchy for S can be built in time $\min\{2^{O(\text{ddim})} n, O(n^2)\}$ (where $n = |S|$), and this term also bounds the space needed to store the hierarchy [13, 19, 26]. (This stored hierarchy is *compressed*, in that points which do not cover any other points in the previous net may be represented implicitly.) Similarly, we can maintain links from each hierarchical point in S_i to all neighbors in net S_i within distance $c \cdot 2^i$, and this increases the space requirement to

$\min\{c^{O(\mathrm{ddim})}n, O(n^2)\}$. From a hierarchy, a *net-tree* may be extracted by placing an edge between each point $p \in S_i$ and its parent in S_{i+1} [26]. The height of the (compressed) tree is bounded by $O(\min\{n, \log \Delta\})$.

Near neighbor problem. A standard technique for ANN on set $V \subset \ell_p$ is the reduction of this problem to that of solving a series of so-called approximate *near* neighbor problems [18,22,27] (also called the Point Location in Equal Balls problem). The c-approximate near neighbor problem for a fixed distance r and parameter $c > 1$ is defined thus:

- If there is a point in V within distance r of query q, return some point in V within distance cr of q.
- If there is no point in V within distance r of query q, return *null* or some point in V within distance cr of q.

For example, suppose we had access to an oracle for the c-approximate near neighbor problem. If we preprocess a series of oracles for the $O(\log \Delta)$ values $r = \{\mathrm{diam}(V), \frac{\mathrm{diam}(V)}{2}, \frac{\mathrm{diam}(V)}{4}, \ldots\}$, and query them all, then clearly one of these queries would return a $2c$-ANN of q. In particular, if r' is the distance from q to its nearest neighbor in V, then the oracle query for the value for r satisfying $r' \leq r < 2r'$ would return such a solution. Further, it suffices to seek the minimum r that returns an answer other then *null*. Then we may execute a binary search over the candidate values of r, and so $O(\log \log \Delta)$ oracle queries suffice.

Har-Peled *et al.* [18] show that for all metric spaces, the ANN problem can be solved by making only $O(\log n)$ queries to oracles for the near neighbor problem. The space requirement is $O(\log^2 n)$ times that required to store a single oracle. The reduction incurs a loss in the approximation factor, but this loss can be made arbitrarily small. In Sect. 3, we will require a more specialized reduction, where we allow near neighbor oracle queries only on problem instances that have constant aspect ratio.

2 ANN for ℓ_p-Space via Embedding into ℓ_∞

In this section, we first review the the algorithm of Andoni (Sect. 2.1), which includes his embedding from ℓ_p-space into ℓ_∞ and utilization of Indyk's ℓ_∞ ANN structure. We then refine this technique in Sect. 2.2 to give distortion that depends on the doubling dimension of the space instead of its cardinality.

2.1 Embedding into ℓ_∞

Here we show that any n-point ℓ_p^d space admits an oblivious embedding into ℓ_∞^d with favorable properties: The embedding is non-contractive with high probability, while the interpoint expansion is small. Hence the embedding approximately preserves the nearest neighbor for a fixed query point q, and keeps more distant points far away. This implies that Indyk's ℓ_∞ ANN algorithm can be applied to all ℓ_p.

Max-stability. The embedding utilizes max-stable random variables, specifically those drawn from a Fréchet distribution: Having fixed p, the Fréchet random variable Z obeys for all $x > 0$, $\Pr[Z \leq x] = e^{-x^{-p}}$. We state the well-known max-stability property of the Fréchet distribution:

Fact 1. *Let random variables $X, Z_1, \ldots, Z_d$ be drawn from the above Fréchet distribution, and let $v = (v_1, \ldots, v_d)$ be a non-negative valued vector. Then the random variable $Y := \max_i\{v_i Z_i\}$ is distributed as $\|v\|_p \cdot X$ (that is, $Y \sim \|v\|_p \cdot X$).*

To see this, observe that $\Pr[Y \leq x] = \Pr[\max_i\{v_i Z_i\} \leq x] = \Pi_i \Pr[v_i Z_i \leq x] = \Pi_i \Pr[Z_i \leq x/v_i] = \Pi_i e^{-(v_i/x)^p} = e^{-(\sum_i v_i^p)/x^p} = e^{-(\|v\|_p/x)^p}$. And similarly, $\Pr[\|v\|_p \cdot X \leq x] = \Pr[X \leq x/\|v\|_p] = e^{-(\|v\|_p/x)^p}$. So indeed the two random variables have the same distribution.

Embedding into ℓ_∞. Given set $V \subset \ell_p$ of d-dimensional vectors, the embedding $f_b : V \to \ell_\infty$ (for any constant $b > 0$) is defined as follows: First, we draw d Fréchet random variables $Z_1, \ldots, Z_d$ from the above distribution. Embedding f_b maps each vector $v \in V$ to vector $f_b(v) = (bv_1Z_1, \ldots, bv_dZ_d)$. The resulting set is $V' \in \ell_\infty$. Clearly, the embedding can be computed in time $O(d)$ per point. We prove the following lemma.

Lemma 2. *For all $p \geq 1$, embedding f_b applied to set $V \subset \ell_p$, for $b = (3 \ln n)^{1/p}$ and $t > 2$, satisfies*

- *Contraction: f_b is non-contractive with probability at least $1 - \frac{1}{n}$.*
- *Expansion: For any pair $u, w \in V$, $\|f_b(u) - f_b(w)\|_\infty \leq \frac{b}{\ln^{1/p} t}\|u - w\|_p$ with probability $\frac{1}{t}$.*

Proof. Consider some vector v with $\|v\|_p = 1$. Then by Fact 1, $\|f_b(v)\|_\infty \sim b\|v\|_p \cdot X = b \cdot X$, where X is a Fréchet random variable drawn from the above distribution. Then $\Pr[\|f_b(v)\|_\infty < 1] = \Pr[b \cdot X < 1] = e^{-(1/b)^{-p}} = \frac{1}{n^3}$. Since the embedding is linear, v may be taken to be any inter-point distance between two vectors in V ($v = \frac{u-w}{\|u-w\|_p}$), and so the probability that *any* inter-point distance decreases is less than $n^2 \cdot \frac{1}{n^3} = \frac{1}{n}$. Also, $\Pr[\|f_b(v)\|_\infty \leq \frac{b}{\ln^{1/p} t}] = \frac{1}{t}$, and so for any vector pair $u, w \in V$ we have $\Pr[\|f_b(u) - f_b(w)\|_\infty \leq \frac{b}{\ln^{1/p} t}\|u - w\|_p] = \frac{1}{t}$.

Indyk's near neighbor structure is given a set $V \in \ell_\infty$ and distance r, and answers $O(\log\log d)$-near neighbor queries for distance r in time $O(d \log n)$ and space $n^{1+\delta}$, where δ is an arbitrarily small constant (that affects the approximation bounds). Combining this structure and Lemma 2, we have:

Corollary 1. *Given set $V \subset \ell_p$ for $p > 2$ and a fixed distance r, and $2 < t < n$, there exists a data structure of size and preprocessing time $n^{1+\delta}$ (for arbitrarily small constant δ) that solves the $O(\log\log d \log_t^{1/p} n)$-approximate near neighbor problem for distance r with query time $O(d \log n)$, and is correct with probability at least $\frac{1}{t} - \frac{1}{n}$.*

Proof. Given a set V and distance r, we preprocess the set by computing the embedding of Lemma 2 for each point. On the resulting set we precompute Indyk's structure for distance $r' = O(\log_t^{1/p} n) \cdot r$. Given a query point q, we embed the query point and query Indyk's structure. The space and runtime follow.

For correctness, by the guarantees of Lemma 2, if $\|q-v\|_p \leq r$ for some point $v \in V$, then under the expansion guarantee of the mapping, their ℓ_∞ distance is at most $\frac{2(3\ln n)^{1/p}}{\ln^{1/p} t} \cdot r \leq r'$, so the structure does not return *null*. On the other hand, if the embedding succeeds then it is non-contractive, and so any returned point must be within ℓ_p distance $O(\log\log d) \cdot r'$ of q. The probability follows from the contraction and expansion guarantees of Lemma 2.

Finally, we use the near neighbor algorithm to solve the ANN problem, which was our ultimate goal:

Theorem 2. *Given set $V \subset \ell_p$ for $p > 2$ and any $2 < t < \frac{n}{2}$, there exists a data structure of size and preprocessing time $n^{1+\delta}$ (for arbitrarily small constant δ) which returns an $O(\log\log d \log_t^{1/p} n)$-ANN in time $O(td\log^2 n \log\log n)$, and is correct with constant probability.*

Proof. We invoke the reduction of Har-Peled *et al.* [18] to reduce ANN to $O(\log n)$ near neighbor queries. We require that all $O(\log n)$ queries succeed with constant probability, hence each near neighbor query must be correct with probability $1 - O\left(\frac{1}{\log n}\right)$. Each near neighbor query is resolved by preprocessing and querying $O(t\log\log n)$ independent structures of Corollary 1. The probability that all these structures fail simultaneously is at most $(1-\frac{1}{t}+\frac{1}{n})^{O(t\log\log n)} \leq (1-\frac{1}{2t})^{O(t\log\log n)} = O\left(\frac{1}{\log n}\right)$, and so at least one is correct with the desired probability. The runtime follows.

The reduction of [18] increases the space usage by a factor of $O(\log^2 n)$, and the additional oracles by a factor of $O(t\log\log n)$, but these increases are subsumed under the constant δ in the exponent.

When $p = \Omega(\log\log n)$, the construction of Andoni recovers the $O(\log\log d)$-approximation guarantees of Indyk's ℓ_∞ structure, previously known to extend only to $p = \Omega(\log d)$).

Comment. For constant values of t, one can achieve a better runtime than implied by Theorem 2: Note that for embedding f_b in Lemma 2, we have that $\|f_b(u) - f_b(w)\|_\infty \leq 2b\|u-w\|_p$ with probability at least $1-2^{-p}$. Then in place of Corollary 1, we have a data structure of size $n^{1+\delta}$ that solves the $O(\log\log d \log^{1/p} n)$-approximate near neighbor problem for distance r with query time $O(d\log n)$, and is correct with probability at least $1-\frac{1}{n}-\frac{1}{2^p}$. An analysis similar to that of Theorem 2 – but using $O(\lceil \log_{2^p}\log n\rceil) = O\left(\left\lceil\frac{\log\log n}{p}\right\rceil\right)$ substructures – gives a data structure of size $n^{1+\delta}$ which returns an $O(\log\log d\log^{1/p} n)$-ANN in time $O(d\log^2 n) \cdot \lceil\frac{\log\log n}{p}\rceil$, and is correct with constant probability.

2.2 Embedding with Distortion Dependent on the Doubling Dimension

Here we give an ANN algorithm whose approximation factor depends on the doubling dimension, instead of the cardinality of the space. We begin with a statement that applies only to nets. Our approach is motivated by a technique for low-dimensional Euclidean embeddings that appeared in [23].

Lemma 3. *Let set $V \subset \ell_p$ have minimum inter-point distance 1, and let $q \in \ell_p$ be any query point. Let $\operatorname{ddim} \geq 2$ be the doubling dimension of $V \cup \{q\}$, and fix any value $c \geq 4$. For all $p \geq 1$, embedding f_1 (of Lemma 2) applied to set $V \cup \{q\}$ satisfies*

- *Contraction: Let $W \subset V$ include all points at distance at least $h = c(4 \operatorname{ddim} \ln(16c \operatorname{ddim}))^{1/p}$ from q. Then $\min_{v \in W} \|f_1(q) - f_1(v)\|_\infty > c$, with probability at least $1 - \operatorname{ddim}^{-\operatorname{ddim}}$.*
- *Expansion: For any pair $v, w \in V \cup \{q\}$ and $t > 1$, $\|f_1(v) - f_1(w)\|_\infty \leq \frac{1}{\ln^{1/p} t}\|v - w\|_p$ with probability $\frac{1}{t}$.*

Proof. Let $W_i \subset W$ include all points with distance to query point q in the range $[2^i, 2^{i+1})$. Since W has minimum inter-point distance 1 and diameter less that 2^{i+1}, Lemma 1 implies that $|W_i| \leq 2^{(i+2)\operatorname{ddim}}$. Let E_i be the bad event that W_i contains any point $v \in W_i$ for which $\|f_1(v) - f_1(q)\|_\infty \leq c$.

Set $j = \log h$, so that all points in W are found in sets W_{j+k} for integral $k \geq 0$. For any point $v \in W_{j+k}$ the probability that the distance from q to v contracts to c or less is $\Pr[\|f_1(v) - f_1(q)\|_\infty \leq c] = e^{-(\|v-q\|_p/c)^p} \leq e^{-(2^{j+k}/c)^p}$. Hence, the probability of bad event E_{j+k} is at most

$$\Pr[E_{j+k}] \leq |W_{j+k}| \cdot e^{-(2^{j+k}/c)^p} = 2^{(j+k+2)\operatorname{ddim}} e^{-(2^{j+k}/c)^p} \leq e^{-(2^{j+k}/c)^p/2}$$
$$< \operatorname{ddim}^{-2\operatorname{ddim} \cdot 2^k}$$

where the penultimate inequality follows from the fact that for all $x \geq h$ we have $(x/c)^p \geq 2\operatorname{ddim}(\log x + 2)$. The probability that any point in W contracts to within distance c of q is at most $\sum_{k=0}^{\infty} E_{j+k} < \sum_{k=0}^{\infty} \operatorname{ddim}^{-2\operatorname{ddim} \cdot 2^k} < \operatorname{ddim}^{-\operatorname{ddim}}$, as claimed. The expansion guarantee follows directly from Fact 1.

As before, Lemma 3 can be used to solve the near neighbor problem:

Corollary 2. *Given set $V \subset \ell_p$ for $p > 2$, a distance r and any $1 < t < \operatorname{ddim}^{\operatorname{ddim}}$, there exists a data structure of size and preprocessing time $n^{1+\delta}$ (for arbitrarily small constant δ) which solves the $O\left(\log\log d \left(\frac{\operatorname{ddim} \log \operatorname{ddim})}{\log t}\right)^{1/p}\right)$-approximate near neighbor problem for distance r in time $O(d \log n)$, and is correct with probability $\frac{1}{t} - \operatorname{ddim}^{-\operatorname{ddim}}$.*

Proof. Given a set V and distance r, we preprocess the set by extracting an r-net, and then scaling down the magnitude of all vectors by r, so that the resulting set has minimum inter-point distance 1. We then compute the embedding of Lemma 3 into ℓ_∞ for each net-vector, and precompute Indyk's ℓ_∞ structure for distance $\frac{2}{\ln^{1/p} t}$. Given a query point q, we scale it down by r, embed it into ℓ_∞ using the same embedding as before, and query Indyk's structure on distance $\frac{2}{\ln^{1/p} t}$. The space and runtime follows.

For correctness, let $\|q - v\|_p \leq r$ for some point $v \in V$. After extracting the net, some net-point w satisfied $\|w - v\|_p \leq r$, and so $\|q - w\|_p \leq 2r$. After scaling, we have $\|q - w\|_p \leq 2$, and after applying the embedding into ℓ_∞, $\|q - w\|_p \leq \frac{2}{\ln^{1/p} t}$ with probability at least $\frac{1}{t}$. In this case Indyk's near neighbor structure must return a point within distance $O(\log\log d)$ of q in the embedded space (that is ℓ_∞). By the contraction guarantees of Lemma 3 (taking $c = \Theta\left(\frac{\log\log d}{\log^{1/p} t}\right)$ and assuming $d = 2^{o(\mathrm{ddim})}$), the distance from the returned point to q in the scaled ℓ_p space is at most $O\left(\log\log d \cdot \left(\frac{\mathrm{ddim}\log \mathrm{ddim}}{\log t}\right)^{1/p}\right)$, and so it is an $O\left(\log\log d \cdot \left(\frac{\mathrm{ddim}\log \mathrm{ddim}}{\log t}\right)^{1/p}\right)$-approximate near neighbor in the origin space.

Similar to the derivation of Theorem 2, we have:

Theorem 3. *Given set $V \subset \ell_p$ for $p > 2$ and any $1 < t < \mathrm{ddim}^{\mathrm{ddim}}$, there exists a data structure of size and preprocessing time $n^{1+\delta}$ (for arbitrarily small constant δ) which returns an $O\left(\log\log d \cdot \left(\frac{\mathrm{ddim}\log \mathrm{ddim}}{\log t}\right)^{1/p}\right)$-ANN in time $O(td\log^2 n \log\log n)$ and is correct with positive constant probability.*

For constant t, we have that when $p = \Omega(\log \mathrm{ddim})$ we recover the $O(\log\log d)$-approximation of Indyk, and this improves upon the $p = \Omega(\log\log n)$ guarantee of the previous section.

3 ANN for ℓ_p-Space via Embedding into ℓ_2

In this section, we show that an embedding from ℓ_p ($p > 2$) into ℓ_2 can be used to derive a $2^{O(p)}$-ANN in logarithmic query time.

We review the guarantees of the Mazur map below, and show it can be used as an embedding into ℓ_2. We then solve the ANN problem in the embedded space. This is however non-trivial, as the map incurs distortion that depends on the set diameter, a problem we address below.

3.1 The Mazur Map

The Mazur map for the real valued vectors is defined as a mapping from ℓ_p^m to ℓ_q^m (for any $0 < p, q < \infty$). The mapping of vector $v \in \ell_p$ is defined as $M(v) = \{|v(0)|^{p/q}, |v(1)|^{p/q}, \ldots, |v(m-1)|^{p/q}\}$, where $v(i)$ is the i-th coordinate of v. The following theorem introduces a scaled Mazur map, and is adapted from [10].

Theorem 4. *Let $x, y \in \ell_p$, $p < \infty$, be vectors such that $\|x\|_p, \|y\|_p \leq C$. The Mazur map for $1 \leq q < p$ scaled down by factor $\frac{p}{q}C^{p/q-1}$ fulfills the following:*

- *Expansion: The mapping is non-expansive.*
- *Contraction: $\|M(x) - M(y)\|_q \geq \frac{q}{p}(2C)^{1-p/q}\|x - y\|_p^{p/q}$.*

The scaled Mazur map implies an embedding from ℓ_p ($p > 2$) into ℓ_2, as in the following. (See also [28, Lemma 7.6], a significantly more general result.)

Corollary 3. *Any subset $V \subset \ell_p$, $p < \infty$ with $\|x\|_p \leq C$ for all $x \in V$ possesses an embedding $f : V \rightarrow l_2$ with the following properties for all $x, y \in V$:*

- *Expansion: The embedding f is non-expansive.*
- *Contraction: For $\|x - y\|_p = u$, $\|f(x) - f(y)\|_q \geq \frac{2}{p}(2C)^{1-p/2}u^{p/2}$.*

3.2 Nearest Neighbor Search via the Mazur Map

Using the Mazur map, we can give an efficient algorithm for ANN for all $p > 2$. Recall that by definition, ddim $= O(\log n)$. First we define the c-bounded near neighbor problem (c-BNN) for $c > 9$ as follows: For a d-dimensional set V for which $\|x\|_p \leq c$ for all $x \in V$, given a query point q:

- If there is a point in V within distance 1 of query q, return some point in V within distance $\frac{c}{9}$ of q.
- If there is no point in V within distance 1 of query q, return *null* or some point in V within distance $\frac{c}{9}$ of q.

(The term $\frac{c}{9}$ was chosen to simplify the calculations below.)

Lemma 4. *For $c = p18^{p/2}$, there exists a data structure for the c-bounded near neighbor problem for $V \subset \ell_p$, $p > 2$, that preprocesses V in time and space $n^{O(1)}$, and resolves a query in time $O(d \log n)$ with probability $1 - n^{-O(1)}$.*

Proof. The points are preprocessed by first applying the scaled Mazur map to embed V into ℓ_2 in time $O(dn)$. We then use the Johnson-Lindenstrauss (JL) transform [24] (or the fast JL transform [1]) to reduce dimension to $d' = O(\log n)$ with no expansion and contraction less than $\frac{1}{2}$, in time $O(dn \log n)$. On the new space, we construct a data structure of size $2^{O(d')} = n^{O(1)}$ supporting Euclidean 2-approximate near neighbor queries in $O(d' \log n)$ time per query [18,27].

Given a query point q, we apply the Mazur map and JL transform on the new point in time $O(d \log n)$, and use the resulting vector as a query for the 2-approximate near neighbor algorithm on the embedded space in time $O(d' \log n)$. If the point x returned by this search satisfies $\|q - x\|_p \leq \frac{c}{9}$ then we return it, and otherwise we return *null*.

To show correctness: The Mazur map is non-expansive, as is the JL transform (which is correct with probability $1 - n^{-O(1)}$). By Corollary 3, the Mazur map ensures that inter-point distances of $\frac{c}{9}$ or greater map to at least

$\frac{2}{p}(2c)^{1-p/2}(c/9)^{p/2} = \frac{2}{p}2c18^{-p/2} = \frac{2}{p}2(p18^{p/2})18^{-p/2} = 4$, and then the contraction guarantee of the JL-transform implies that the distance in the embedded Euclidean space is greater than 2. It follows that if q possesses a neighbor in the original space at distance 1 or less, the 2-ANN in the embedded Euclidean space finds a neighbor at distance 2 in the embedded space and less than $\frac{c}{9}$ in the origin space.

We will show that an oracle solving c-BNN can be used as a subroutine for a data structure solving the c-approximate nearest neighbor problem. This parallels the classical reduction from ANN to the near neighbor problem utilized above. However, in order to minimize the distortion introduced by the Mazur map we must restrict the oracle to bounded diameter sets, and this results in a different reduction, adapted from [26].

Theorem 5. *Let V and c be as in Lemma 4. There exists a data structure for the $6c = 6p18^{p/2}$-ANN problem on V, which preprocesses $\min\{2^{O(\text{ddim})}n, O(n^2)\} \cdot \left\lceil \frac{\log\log d}{p\,\text{ddim}} \right\rceil$ separate c-BNN oracles, each for a subset of V of size at most $z = \min\{c^{O(\text{ddim}(S))}, n\}$, in total time and space $O(znd\log\log d)$. The structure resolves a query with constant probability or correctness by executing $O\left(\log d \cdot \left\lceil \frac{\log\log d}{\log z} \right\rceil\right)$ c-BNN oracle invocations, in total time $O\left(d^2\log n + d\log d \cdot \log z \cdot \left\lceil \frac{\log\log d}{\log z} \right\rceil\right)$.*

Proof. We preprocess a hierarchy and net-tree for V. Given query point q, we will seek a hierarchical point $w \in S_j$ which satisfies $\|w - q\|_p \le 3 \cdot 2^j$, for minimal j. Note that if such a point w exists, then every hierarchical ancestor $w' \in S_i$ $(i > j)$ of w also satisfies $\|w' - q\|_p \le 3 \cdot 2^i$: We have $\|w' - w\|_p \le 2 \cdot 2^i - 2 \cdot 2^j$, and so $\|w' - q\|_p \le \|w' - w\|_p + \|w - q\|_p < (2 \cdot 2^i - 2 \cdot 2^j) + 3 \cdot 2^j < 3 \cdot 2^i$.

To find w, we modify the navigating-net algorithm of Krauthgamer and Lee [26]: Beginning with the root point at the top level of the hierarchy, we descend down the levels of the hierarchy, while maintaining at each level S_i a single point of interest $t \in S_i$ satisfying $d(t, q) \le 3 \cdot 2^i$.

Let $t \in S_i$ be the point of interest in level S_i. Then the next point of interest $t' \in S_{i-1}$ satisfies $\|t' - t\|_p \le \|t' - q\|_p + \|q - t\|_p \le 3 \cdot 2^i + 3 \cdot 2^{i-1} = 4.5 \cdot 2^i$. So to find t' it suffices to search all points of S_{i-1} within distance $4.5 \cdot 2^i$ of t. But there may be $2^{\Theta(\text{ddim})}$ such points, and so we cannot afford a brute-force search on the set. Instead, we utilize the c-BNN data structure of Lemma 4: For each net-point $t \in S_i$, preprocess a set $N(t, i)$ that includes all these candidate points of S_{i-1}, as well as all their descendants in the hierarchical level S_k, $k = i - \lceil \log 4c \rceil$ within distance $4.5 \cdot 2^i$ of t. After translating $N(t, i)$ so that t is the origin, and scaling so that all points have magnitude at most c, we preprocess for $N(t, i)$ the c-BNN oracle of Lemma 4. (Note that $|N(t, i)| = O(w)$.) To find t', execute a c-BNN query on $N(t, i)$ and q, and if the query returns a point q', then set t' to be the ancestor of q' in S_{i-1}: $\|t' - q\|_p \le \|t' - q'\|_p + \|q' - q\|_p < 2 \cdot 2^{i-1} + \frac{4.5 \cdot 2^i}{9} = 3 \cdot 2^{i-1}$.

We terminate the algorithm in any of three events:

– The root $t \in S_i$ does not satisfy $\|q - t\|_p \leq 3 \cdot 2^i$. In this case the root itself is at worst a 3-ANN of q, as the distance from all descendants of the root (that is, all points) to q is greater than $\|q - t\|_p - 2 \cdot 2^i$.
– The algorithm locates a point $w \in S_0$ satisfying $\|w - q\| \leq 3$. Then the nearest neighbor of q can be either w or a point within distance 6 of w. We execute a c-BNN query on the set of points within distance 6 of w. (That is, as above we translate w to the origin, scale so that the maximum magnitude is c, and preprocess a query structure.) If the query returns *null*, then there is no point within distance $\frac{6}{c}$ of q, and so w is a $\frac{3}{6/c} = \frac{c}{2}$-ANN of q. If the query return some point, then that point is within distance $\frac{6}{9} = \frac{2}{3}$ of q. Since the minimum inter-point distance of the set is 1, there cannot be another point within distance $\frac{1}{3}$ of q, and so the returned point is a 2-ANN of q.
– A c-BNN query on some set $N(t, i)$ returns *null*. As the diameter of $N(t, i)$ is at least 2^i, this implies that there is no point in $N(t, i)$ within distance $\frac{2^i}{c}$ of q. As all descendants of a point in S_k ($k = i - \lceil \log 4c \rceil$) are within distance $2 \cdot 2^k$ of their ancestor, the distance from q to all other points is at least $\frac{2^i}{c} - 2 \cdot 2^k = \frac{2^i}{c} - 2\frac{2^i}{4c} = \frac{2^i}{2c}$. It follows that t is a $\frac{3 \cdot 2^i}{2^i/2c} = 6c$-ANN of q.

We conclude that the above algorithm returns the nearest neighbor stipulated by the lemma. However, its runtime depends on the number of levels in the hierarchy, that is $O(\min\{n, \Delta\})$. To remove this dependence, we first invoke the algorithm of Chan [11] to find in time $O(d^2 \log n)$ and high probability a $O(d^{3/2})$-ANN of q, called q'. We then locate the ancestor of q' in level S_i, $i = \lceil \log \|q - q'\|_p \rceil$, of the hierarchy in time $O(\log n)$, which can be done easily using standard tree decomposition algorithms. We assign this ancestor as our first point of interest t; note that we have $\|t - q\|_p \leq \|t - q'\|_p + \|q' - q\|_p < 2 \cdot 2^i + 2^i = 3 \cdot 2^i$, so indeed t is a valid point of interest. After descending $O(\log d)$ levels in the search, we reach radii that are smaller than the true distance from q to its nearest neighbor in V, and the search must terminate.

In order for the entire procedure to succeed with constant probability, we require the failure probability of each level c-BNN query to be $O(1/\log d)$. Each c-BNN oracle consists of $O\left(\left\lceil \frac{\log \log d}{\log z} \right\rceil\right)$ structures of Lemma 4, and (recalling that a query is executed on a set of size $O(z)$) the probability that they all fail simultaneously is $2^{-O(\log z \cdot (\log \log d)/\log z)} = O(1/\log d)$. The final space and runtime follow directly from the time and space required for building and querying the c-BNN oracles, each of size $O(z)$, plus the single level ancestor query.

Comment. Note that the query time is linear in the doubling dimension, as opposed to the exponential dependence common to ANN for doubling metrics. We can extend this lemma by noting that once a $2^{O(p)}$-ANN is found, we can run the standard navigating net algorithm to descend $O(p)$ additional levels and locate a constant-factor ANN in time $p2^{O(\mathrm{ddim})}$. We then search $\varepsilon^{-O(\mathrm{ddim})}$ more points in a brute-force fashion and locate a $(1 + \varepsilon)$-ANN. So a $(1 + \varepsilon)$-ANN can be found with $p2^{O(\mathrm{ddim})} + \varepsilon^{-O(\mathrm{ddim})}$ additional work.

Acknowledgements. We thank Sariel Har-Peled, Piotr Indyk, Robi Krauthgamer, Assaf Naor and Gideon Schechtman for helpful conversations.

References

1. Ailon, N., Chazelle, B.: Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In: STOC 2006, pp. 557–563 (2006)
2. Andoni, A., Croitoru, D., Patrascu, M.: Hardness of nearest neighbor under L-infinity. In: Foundations of Computer Science, pp. 424–433 (2008)
3. Andoni, A.: Nearest neighbor search: the old, the new, and the impossible. Ph.D. thesis, MIT (2009)
4. Arya, S., Malamatos, T.: Linear-size approximate Voronoi diagrams. In: SODA 2002, pp. 147–155 (2002)
5. Ball, K.: Isometric embedding in l_p-spaces. Eur. J. Comb. **11**(4), 305–311 (1990)
6. Batu, T., Ergun, F., Sahinalp, C.: Oblivious string embeddings and edit distance approximations. In: SODA 2006, pp. 792–801 (2006)
7. Bellman, R.E.: Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton (1961)
8. Bern, M.: Approximate closest-point queries in high dimensions. Inf. Process. Lett. **45**(2), 95–99 (1993)
9. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: ICML 2006, pp. 97–104 (2006)
10. Binyamini, Y., Lindenstrauss, J.: Geometric Nonlinear Functional Analysis. Colloquium Publications (American Mathematical Society), Providence (2000)
11. Chan, T.M.: Approximate nearest neighbor queries revisited. Discret. Comput. Geom. **20**(3), 359–373 (1998)
12. Clarkson, K.L.: A randomized algorithm for closest-point queries. SIAM J. Comput. **17**(4), 830–847 (1988)
13. Cole, R., Gottlieb, L.: Searching dynamic point sets in spaces with bounded doubling dimension. In: STOC 2006, pp. 574–583 (2006)
14. de Amorim, R.C., Mirkin, B.: Minkowski metric feature weighting and anomalous cluster initializing in k-means clustering. Pattern Recogn. **45**(3), 1061–1075 (2012)
15. Fichet, B.: l_p-spaces in data analysis. In: Bock, H.H. (ed.) Classification and Related Metods of Data Analysis, pp. 439–444. North-Holland, Amsterdam (1988)
16. Finlayson, G.D., Rey, P.A.T., Trezzi, E.: General l_p constrained approach for colour constancy. In: ICCV Workshops 2011, pp. 790–797 (2011)
17. Finlayson, G.D., Trezzi, E.: Shades of gray and colour constancy. In: CIC 2004, pp. 37–41 (2004)
18. Har-Peled, S., Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. Theory Comput. **8**(1), 321–350 (2012)
19. Har-Peled, S., Mendel, M.: Fast construction of nets in low-dimensional metrics and their applications. SIAM J. Comput. **35**(5), 1148–1184 (2006)
20. Har-Peled, S., Kumar, N.: Approximating minimization diagrams and generalized proximity search. SIAM J. Comput. **44**(4), 944–974 (2015)
21. Indyk, P.: On approximate nearest neighbors in non-Euclidean spaces. In: FOCS, pp. 148–155 (1998)
22. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC 1998, pp. 604–613 (1998)
23. Indyk, P., Naor, A.: Nearest-neighbor-preserving embeddings. ACM Trans. Algorithms **3**(3), 31 (2007)

24. Johnson, W.B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. In: Conference in Modern Analysis and Probability, New Haven, Connecticut, 1982, pp. 189–206. American Mathematical Society, Providence (1984)
25. Johnson, W.B., Schechtman, G.: Embedding l_p^m into l_1^n. Acta Math. **149**(1–2), 71–85 (1982)
26. Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: SODA 2004, pp. 791–801 (2004)
27. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. In: STOC 1998, pp. 614–623 (1998)
28. Naor, A.: Comparison of metric spectral gaps. Anal. Geome. Metr. Spaces **2**(1), 1–52 (2014)
29. Naor, A., Rabani, Y.: On approximate nearest neighbor search in ℓ_p, $p > 2$ (2006, manuscript)
30. Neylon, T.: A locality-sensitive hash for real vectors. In: SODA 2010, pp. 1179–1189 (2010)
31. Ostrovsky, R., Rabani, Y.: Polynomial time approximation schemes for geometric k-clustering. In: FOCS 2000, pp. 349–358 (2000)
32. Ostrovsky, R., Rabani, Y.: Polynomial-time approximation schemes for geometric min-sum median clustering. J. ACM **49**(2), 139–156 (2002)
33. Yu, D., Yu, X., Wu, A.: Making the nearest neighbor meaningful for time series classification. In: CISP 2011, pp. 2481–2485 (2011)

The Impact of Locality on the Detection of Cycles in the Broadcast Congested Clique Model

Florent Becker[1], Pedro Montealegre[2](✉), Ivan Rapaport[3], and Ioan Todinca[1]

[1] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans, France
[2] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile
p.montealegre@uai.cl
[3] DIM-CMM (UMI 2807 CNRS), Universidad de Chile, Santiago, Chile

Abstract. The *broadcast congested clique* model is a message-passing model of distributed computation where n nodes communicate with each other in synchronous rounds. The joint input to the n nodes is an undirected graph G on the same set of nodes, with each node receiving the list of its *immediate neighbors in* G. In each round each node sends *the same message* to all other nodes, depending on its own input, on the messages it has received so far, and on a public sequence of random bits. One parameter of this model is an upper bound b on the size of the messages, also known as bandwidth. In this paper we introduce another parameter to the model. We study the situation where the nodes, initially, instead of knowing their immediate neighbors, know their neighborhood up to a fixed radius r.

In this new framework we study one of the hardest problems in distributed graph algorithms, this is, the problem of detecting, in G, an induced cycle of length at most k ($\text{Cycle}_{\leq k}$) and the problem of detecting in G an induced cycle of length at least $\text{k}+1$ ($\text{Cycle}_{>k}$). For $r = 1$, we exhibit a deterministic, one-round algorithm for solving $\text{Cycle}_{\leq k}$ with $b = \mathcal{O}(n^{2/k} \log n)$ if k is even, and $b = \mathcal{O}(n^{2/(k-1)} \log n)$ if k is odd. We also prove, assuming the Erdős Girth Conjecture, that this result is tight for $k \geq 4$, up to logarithmic factors. More precisely, if each node, instead of being able to see only its immediate neighbors, is allowed to see up to distance $r = \lfloor k/4 \rfloor$, and if we also allowed randomization and multiple rounds, then the number of rounds R needed to solve $\text{Cycle}_{\leq k}$ must be such that $R \cdot b = \Omega(n^{2/k})$ if k is even, and $R \cdot b = \Omega(n^{2/(k-1)})$ if k is odd.

On the other hand, we show that, if each node is allowed to see up to distance $r = \lfloor k/2 \rfloor + 1$, then a polylogarithmic bandwidth is sufficient for solving $\text{Cycle}_{>k}$ with only two rounds. Nevertheless, if nodes were allowed to see up to distance $r = \lfloor k/3 \rfloor$, then any one-round algorithm that solves $\text{Cycle}_{>k}$ needs the bandwidth b to be at least $\Omega(n/\log n)$.

Keywords: Broadcast congested clique · Induced cycles
Graph degeneracy

Additional support from CONICYT via Basal in Applied Mathematics (P.M. and I.R.) and Fondecyt 1170021 (I.R.).

1 Introduction

The *broadcast congested clique* model is a message-passing model of distributed computation where n nodes communicate with each other in synchronous rounds over a *complete network* [1,2,4,6–8,14,16,18,21]. The joint input to the n nodes is an undirected graph G on the same set of nodes, with node u receiving the list of its neighbors in G. Nodes have pairwise distinct identities, which are numbers upper bounded by some polynomial in n. The identity of node u is denoted by $\mathrm{id}(u)$. All nodes know n, the size of the network.

Each node broadcasts, in each round, a single b-bit message along each of its $n-1$ communication links. The size of the messages is known as the bandwidth of the system, and it is a parameter of the model (which could grow with n). Broadcasting is equivalent to writing the messages on a whiteboard, visible to every node. In each round every node produces its message using its input, the contents of the whiteboard, and a sequence of public random bits.

Typically, the goal of an algorithm is to decide whether the input graph G belongs to some graph class $\mathcal{C}$. An algorithm is correct if it terminates with every node knowing the correct answer (that is, whether $G \in \mathcal{C}$) with high probability. The round complexity of an algorithm is the maximum number of rounds over all possible input graphs (of size n).

Few fast algorithms are known in the broadcast congested clique model. In fact, if the bandwidth $b = \mathcal{O}(\log n)$, then there exist one-round algorithms for deciding whether the input graph G has bounded degeneracy [6], contains a fixed forest [14], is a cograph [21]. Also, if $b = \mathcal{O}(\mathrm{polylog} n)$, then there is a one-round algorithm for deciding whether G is connected [1,2].

One way to increase the computing power of the model is to lift the broadcast restriction and to allow the nodes the possibility of sending *different* messages through different links. This general model, known as *unicast congested clique* [14], gives the possibility to perform a load balancing procedure efficiently. Such enormous intrinsic power has allowed some authors to provide fast algorithms for solving natural problems: an $\mathcal{O}(\log\log\log n)$-round algorithm for finding a 3-ruling set [17], $\mathcal{O}(n^{0.158})$-round algorithms for counting triangles, for counting 4-cycles and for computing the girth [12], an $\mathcal{O}(1)$-round algorithm for detecting a 4-cycle [12], an $\mathcal{O}(1)$-round algorithm constructing a minimum spanning tree [20].

Another very natural, much more limited and less dramatic way to increase the computing power of the broadcast congested clique model, *is to expand the local knowledge the nodes initially have about* G. The idea of a constant-radius neighborhood independent of the size of the network is present in the research on local algorithms pioneered by Angluin [3], Linial [23] and Naor and Stockmeyer [25].

We therefore use the KT_r notion, introduced by Awebuch et al. [5], which means *Knowledge of Topology* up to distance r, excluding edges with both endpoints at distance r. More precisely, we call BClique$[r]$ the extension of the broadcast congested clique model where each node u “sees” (receives as input) the set of **all edges** lying on a path of length at most r, starting in u. Hence, BClique[1] corresponds to the classical broadcast congested clique model, and is simply denoted BClique.

One of the most studied problems in the BClique model is related to the existence of cycles in the input graph G. The first natural question one can formulate, that is, deciding whether G contains a cycle has been, until now, the only question amenable to a simple algorithm. In fact, Becker et al. [6] show that a simple set of logarithmic size messages is sufficient to recognize, deterministically and in one round, whether the input graph G is acyclic.

Any other natural question concerning cycles has given strong negative results. Drucker et al. [14] showed that, if $\ell \geq 4$, then any algorithm that decides whether the ℓ-node cycle C_ℓ is a subgraph (or an induced subgraph) of the input graph G needs $\Omega(ex(n, C_\ell)/nb)$ rounds, where $ex(n, H)$ is the Turán number of H, i.e., the maximal number of edges of an n-node graph which does not contain a subgraph isomorphic to H. Remark that $ex(n, C_\ell)$ is $\Theta(n^2)$ for odd values ℓ, and $\Theta(n^{1+1/\ell})$ for even values (assuming the Erdős Girth Conjecture[1] [15]).

Moreover, *even in the very powerful unicast congested clique model*, the algorithms for cycle detection are rather slow. In fact, the best algorithm for detecting C_ℓ uses $\mathcal{O}(n^\rho \log n)$ rounds, for every $\ell \geq 3$, where $\rho < 0.15715$ [12]. The only exception being the detection of squares C_4, for which an extremely elegant $\mathcal{O}(1)$-round algorithm has been devised [12].

In this paper, we mainly study two problems: Cycle$_{\leq k}$ and Cycle$_{>k}$. The first one consists in deciding whether the graph contains an induced cycle of length at most k (i.e., deciding whether the *girth* of the graph is at most k). The second problem, complementary to the first one, consists in detecting the existence of an induced cycle of length at least $k + 1$. This difficulty to find fast algorithms for problems related to the existence of cycles is what makes the positive results of this paper surprising.

Note that the existence of an induced cycle of length at most k is equivalent to the existence of a cycle (not necessarily induced) of length at most k. On the other hand, as we are going to explain later, finding an algorithm for detecting induced cycles of length at least $k + 1$ requires much more involved arguments than finding algorithms for detecting cycles (not necessarily induced) of length at least $k + 1$.

Our Results

In Sect. 3 we show that there is a deterministic, one-round BClique algorithm for solving problem Cycle$_{\leq k}$ with bandwidth $\mathcal{O}(n^{2/k} \log n)$ if k is even, and bandwidth $\mathcal{O}(n^{2/(k-1)} \log n)$ if k is odd. The main ingredient for proving this is a deterministic, one-round algorithm given in [24] that reconstructs a graph of degeneracy at most d in the BClique model using bandwidth $\mathcal{O}(d \log n)$ (reconstruction means that every node knows all the edges of the input graph).

[1] This conjecture states that there exist graphs with n vertices and $\Omega(n^{1+1/k})$ edges not containing cycles of length less than or equal to $2k$.

Recall that the degeneracy of G is the minimum d such that, by iteratively removing vertices of degree at most d, we obtain the empty graph.

We also show that previous upper bounds match the lower bounds up to logarithmic factors, even in the BCLIQUE$[\lfloor k/4 \rfloor]$ model allowing randomization and multiple rounds. More precisely, if we allowed the nodes to see up to distance $\lfloor k/4 \rfloor$, to use public coins and multiple rounds, then the number of rounds R and bandwidth b needed to solve CYCLE$_{\leq k}$ is such that $R \cdot b = \Omega(n^{2/k})$ if k is even, and $R \cdot b = \Omega(n^{2/(k-1)})$ if k is odd (in both cases $k \geq 4$), for every ϵ-error algorithm. (For these lower bounds we assume the Erdős Girth Conjecture).

We start Sect. 4 by giving a useful, "local" characterization of graphs which do not have long induced cycles. Using this, together with a technique inspired by the linear sketches of [1,19], we show that, if each node is allowed to see at distance $\lfloor k/2 \rfloor + 1$, then a polylogarithmic number of bits is sufficient for detecting in two rounds an induced cycle of length strictly larger than k. More precisely, we prove that for every $k \geq 3$, there exists a two-round algorithm in the BCLIQUE$[\lfloor k/2 \rfloor + 1]$ model that solves CYCLE$_{>k}$ with high probability using bandwidth $\mathcal{O}(\log^4 n)$. The approach is based on the randomized algorithm of Ahn *et al.* [1] for computing a spanning forest in the BCLIQUE model with bandwidth $\mathcal{O}(\log^3 n)$. With respect to lower bounds, we prove that any one-round, public-coin BCLIQUE$[\lfloor k/3 \rfloor]$ algorithm that solves CYCLE$_{>k}$ needs the bandwidth to be at least $\Omega(n/\log n)$. Note that the case $k = 3$ corresponds to decide whether the input graph G is chordal, i.e., whether the only induced cycles in G are triangles.

The results of this article are summarized in Tables 1 and 2.

Table 1. Results concerning problem CYCLE$_{\leq k}$. The lower bounds assume the Erdős Girth Conjecture.

	BCLIQUE$[r]$	#Rounds	Bandwidth	Randomized?
Upper bound Theorem	$r = 1$	1	$O(n^{2/k} \log n)$, k even $O(n^{2/(k-1)} \log n)$, k odd	Deterministic
Lower bound Theorem	$r \leq k/3$	1	$\Omega(n^{2/k} / \log n)$, k even $\Omega(n^{2/(k-1)} / \log n)$, k odd	Randomized ϵ-error
Lower bound Theorem	$r \leq k/4$	R	$\Omega(n^{2/k} / R)$, k even $\Omega(n^{2/(k-1)} / R)$, k odd	Randomized ϵ-error

Table 2. Results concerning problem CYCLE$_{>k}$

	BCLIQUE$[r]$	#Rounds	Bandwidth	Randomized?
Upper bound Theorem	$r \geq \frac{k}{2} + 1$	2	$O(\log^4 n)$	Randomized (w.h.p.)
Lower bound Theorem	$r \leq k/3$	1	$\Omega(n/\log n)$	Randomized ϵ-error

2 Basic Definitions and Notations

Let $G = (V, E)$ be an undirected graph, and let $u \in V$. We call $N_G(u) = \{v \in V | uv \in E\}$ and $N_G[u] = N_G(u) \cup \{u\}$, the *open and closed neighborhoods* of u, respectively. Similarly, for $U \subseteq V$, $N_G(U) = \cup_{u \in U} N_G(u) - U$ and $N_G[U] = N_G(U) \cup \{U\}$ are the *open and closed neighborhoods* of U, respectively. When no ambiguity is possible, we will omit the subindices. By extension, we denote $N^r[u]$ the set of vertices at distance at most r from u, and we call it *closed* r*-neighborhood* of u. Analogously, $N^r(u) = N^r[u] \setminus \{u\}$ is the *open* r*-neighborhood* of u.

Graph $H = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If, for any edge $uv \in E$ with $u, v \in V'$ we also have $uv \in E'$, we say that H is an *induced* subgraph of G, or that H is the subgraph of G induced by V'. Given a vertex subset S, the subgraph induced by S is denoted $G[S]$. We simply write $G - S$ for $G[V \setminus S]$. Also, if F is a subset of edges, we denote by $G - F$ the graph obtained from G by removing the edges of F. The degeneracy of a graph G is the minimum d such that, by iteratively removing vertices of degree at most d, we obtain the empty graph.

If S is a vertex subset of $G = (V, E)$, the *contraction* of S consists in replacing the whole subset S by a unique vertex v_S, such that the neighborhood of v_S in the new graph is $N_G(S)$ while $G - S$ remains unchanged. A *connected component* of G is the inclusion-maximal set of vertices inducing a connected graph. An induced path (resp. cycle) of graph G is called a *chordless* path (cycle). A graph is called k*-chordal* if it does not contain any induced cycle of length greater than k. The 3-chordal graphs are known as *chordal graphs*.

The BClique[r] model is formally defined as follows. There are n nodes which are given distinct identities (IDs), that we assume for simplicity to be numbers between 1 and n. In this paper we consider the situation where the joint input to the nodes *is a graph* G. More precisely, *each node* u *receives as input the subgraph of radius* r *around itself* (i.e., all edges lying on a path of length at most r, starting in u). Nodes execute an algorithm, broadcasting b-bit messages in synchronous rounds. Their goal is to compute some function f that depends on G. When an algorithm stops *every node must know* $f(G)$. Function f defines the problem to be solved. A $0 - 1$ function corresponds to a decision problem.

An algorithm may be deterministic or randomized. We distinguish two subcases of randomized algorithms: the private-coin setting, where each node flips its own coin; and the public-coin setting, where the coin is shared between all nodes. (In this work we are going to consider public-coin algorithms only). An ε-error algorithm $\mathcal{A}$ that computes a function f is a randomized algorithm such that, for every input graph G, $\Pr\{\mathcal{A}$ outputs $f(G)\} \geq 1 - \varepsilon$. In the case where $\varepsilon \to 0$ as $n \to \infty$, we say that $\mathcal{A}$ computes f with high probability (whp).

We consider several decision problems in this paper: Cycle$_{=k}$, Cycle$_{\leq k}$ and Cycle$_{>k}$. These problems consist in deciding, respectively, whether the input graph has an induced cycle of length exactly k, at most k, and strictly larger than k. Problems Sub-Cycle$_{=k}$, Sub-Cycle$_{\leq k}$ and Sub-Cycle$_{>k}$ are defined in a similar way, but in this case we ask whether the input graph has a cycle as a subgraph (induced or not) of length k, at most k, and strictly larger than k.

3 Detection of Short Cycles

Let us denote by $ex(n,k)$ the maximum number of edges in an n-vertex graph not containing a cycle of length at most k. A very helpful result in the study of graphs without short cycles is the one that relates the nonexistence of short cycles in G with the degeneracy of G. More precisely, graphs with no cycles of length at most k (as subgraphs) have a relatively small degeneracy.

Proposition 1 **([14]).** *Graphs with no cycles of length at most k are of degeneracy $\mathcal{O}(ex(n,k)/n)$.*

In [24] it is shown that graphs of degeneracy at most d can be recognized, and even reconstructed, by a one-round algorithm in the BClique model using bandwidth $\mathcal{O}(d \cdot \log n)$. Recall that reconstruction means that at the end of the algorithm, every node knows all the edges of the input graph.

Theorem 1 **([24]).** *There is a one-round, deterministic algorithm in the model* BClique*, that reconstructs the input graph G if the graph is d-degenerate, and rejects otherwise, using bandwidth $\mathcal{O}(d \cdot \log n)$.*

By Proposition 1, the degeneracy of the NO-instances of $\text{Cycle}_{\leq k}$ is upper bounded by $\mathcal{O}(ex(n,k)/n)$. Therefore, from Theorem 1, we conclude the existence of a one-round algorithm for $\text{Cycle}_{\leq k}$ such that, each node, either (1) fully reconstructs the graph and decides the existence of a cycle of length at most k or (2) notices that the degeneracy of the input graph is larger than the bound required by the NO instances, and concludes that the input graph must be a YES instance. Therefore, we have the following corollary.

Corollary 1. *Problem $\text{Cycle}_{\leq k}$ can be solved with a one-round, deterministic algorithm in the* BClique *model using bandwidth $\mathcal{O}((ex(n,k)/n) \log n)$.*

Previous algorithm is rather restrictive. It is deterministic, it works in one-round and the information each node has about the graph is minimal, consisting in the 1-neighborhood. The question we ask here is the following: is it possible, by lifting previous restrictions, to decrease the *total* number of bits broadcasted by each node? Next results give a negative answer to this question. In other words, the one-round deterministic algorithm based on the degeneracy seems to be the best we can do.

Recall that BClique$[r]$ is the extension of the broadcast congested clique model where each node u receives as input the set of all edges lying on a path of length at most r, starting in u. Our first result tackles the case where $r \leq \lfloor k/4 \rfloor$.

Theorem 2. *Let $\epsilon \leq 1/3$ and $0 < r \leq k/4$. Then, any ϵ-error, R-round, b-bandwidth algorithm in the* BClique$[r]$ *model solving $\text{Cycle}_{\leq k}$ satisfies $R \cdot b = \Omega(ex(n,k)/n)$.*

In the case where the nodes have more knowledge of the graph, i.e., when $k/4 \leq r \leq k/3$, we obtain a tight bound for one-round algorithms.

Theorem 3. *Let $\epsilon \leq 1/3$ and $k/4 < r \leq k/3$. Then, any ϵ-error, one-round algorithm in the* BCLIQUE$[r]$ *model that solves* CYCLE$_{\leq k}$ *requires bandwidth $b = \Omega(ex(n,k)/(n \log n))$.*

Remark 1. Bondy and Simonovits [10] showed that $ex(n,k) = \mathcal{O}(n^{1+2/k})$ if k is even, and $ex(n,k) = \mathcal{O}(n^{1+2/(k-1)})$ if k is odd. On the other hand, the Erdős Girth Conjecture states that this bound is tight, implying the results of Table 1. Note that currently, the best constructions provide a lower bound for $ex(n,k) = \Omega(n^{1+4/(3k-7)})$ if k is even, and $ex(n,k) = \Omega(n^{1+4/(3k-9)})$ if k is odd [22].

4 Detection of Long Cycles

Recall that graphs without induced cycles of length greater than k are called k-chordal [11]. 3-chordal graphs, i.e., graphs in which every cycle (not necessarily induced) of 4 or more vertices has a chord, are called chordal graphs. It is known that a graph G is chordal if and only if, for each vertex $u \in V$, and each connected component C in $G - N[u]$, the neighborhood $N(C)$ of this component induces a clique in G. This "local" characterization has been exploited by Chandrasekharan and Sitharama Iyengar [13] for devising a fast parallel algorithm recognizing chordal graphs. We begin this section by extending previous characterization to arbitrary chordalities $k > 3$ in order to take advantage of this in our distributed framework.

Let G be a graph, $u \in V(G)$ and $k > 0$. Let $D_1, \ldots, D_p$ be the p connected components of $G - N^{\lfloor k/2 \rfloor}[u]$ (obtained by removing the vertices at distance at most $\lfloor k/2 \rfloor$ from u). Let H_u^k denote the graph obtained from G by contracting each component D_i into a single node d_i.

Lemma 1. *Let G be a graph. G is k-chordal if and only if, for every $u \in V(G)$, H_u^k is k-chordal.*

Lemma 1 provides us with a strategy for deciding k-chordality, i.e., for deciding whether the input graph G is a NO instance of problem CYCLE$_{>k}$. For doing this every node x must compute the graph H_x^k and then decides whether H_x^k is k-chordal. In order to compute H_x^k, each node x needs first to find the connected components of $G - N^{\lfloor k/2 \rfloor}[x]$. Let F_x is the set of all edges lying on a path of length at most $\lfloor k/2 \rfloor + 1$ starting in x. We need then each node to compute the connected components of $G - F_x$ outside $N^{\lfloor k/2 \rfloor}[x]$.

4.1 Computing the Connected Components of $G - F_x$

Ahn et al. provide a probabilistic, one-round algorithm for computing a spanning forest of the input graph G, in the BCLIQUE model using bandwidth $\mathcal{O}(\log^3 n)$ [1]. In their algorithm, each node constructs a message based on its neighborhood and on a sequence of public random coins, and broadcasts it to all other nodes. Using all these messages, every node is able to construct a spanning forest of the graph with probability $1 - \epsilon$, for a fixed $\epsilon > 0$.

We want each node x to compute the connected components of $G - F_x$. Recall that F_x is the set of all edges lying on a path of length at most $\lfloor k/2 \rfloor + 1$ starting in x. We place ourselves in the BClique[$\lfloor k/2 \rfloor + 1$] model with bandwidth $\mathcal{O}(\log^4 n)$. We amplify the bandwidth by a $\log(n)$ factor, with respect to the spanning tree algorithm of [1], to ensure that it succeeds with high probability. Also, every node needs to know all the set of edges F_x, that is why we choose the BClique[$\lfloor k/2 \rfloor + 1$] model. Using the spanning forest algorithm of [1], we prove that each node x can construct a spanning forest of $G - F_x$ with high probability.

The key observation is that the messages produced by each vertex is a linear function (w.r.t. to the edges of the graph). Therefore, from the messages of G, each vertex x computes the messages that the algorithm *would have constructed* on $G - F_x$.

Definition 1. *Let $n, k, \delta > 0$. A δ-linear sketch of size k is a function S: $\{0,1\}^{\mathcal{O}(\log n)} \times \{-1,0,1\}^n \to \{0,1\}^k$, such that, if we call $S_r = S(r, \cdot)$, then*

- *S_r is linear, for each $r \in \{0,1\}^{\mathcal{O}(\log n)}$;*
- *If r is chosen uniformly at random, then there is an algorithm that on input $S_r(x)$ returns* ERROR *with probability at most δ, and otherwise returns a pair (i, x_i) such that $x_i \neq 0$ and coordinate i is picked uniformly at random between the non-zero coordinates of x. The probabilities are taken over the random choices of r.*

Proposition 2 ([19]). *For each $n, \delta > 0$, there exists a δ-linear sketch of size $\mathcal{O}(\log^2 n \log \delta^{-1})$.*

Let $G = (V, E)$ be a graph of size n, and $x \in V$. We call a^x the *connectivity vector of x in G*, defined as the vector of dimension $\binom{V}{2}$ such that:

$$a^x_{\{u,v\}} = \begin{cases} 1 & \text{if } \{u,v\} \in E, x = u \text{ and } u < v, \\ -1 & \text{if } \{u,v\} \in E, x = v \text{ and } u < v, \\ 0 & \text{otherwise.} \end{cases}$$

For $r \in \{0,1\}^{\mathcal{O}(\log n)}$, we say that $S_r(G) = \{S_r(a^x)\}_{x \in V(G)}$ is a δ-*connectivity sketch* of G, where S is a δ-linear sketch. Note that for any $x \in V$, each non zero coordinate of a^x represents an edge of $N(x)$, and for any $U \subseteq V$ the non zero coordinates of $\sum_{x \in U} a^x$ are exactly the edges in the cut between U and its complement $V \backslash U$.

Let $G = (V, E)$ be the input graph. The one-round algorithm in the BClique model devised by Ahn et al. for computing a spanning forest of G works as follows. Let $t = \lceil \log n \rceil$. Each node computes and sends t independent δ-linear sketches of its connectivity vector, using t random strings $r_1, \ldots, r_t$ picked uniformly at random. Using these messages, any node can compute t independent δ-connectivity sketches of G and therefore it can compute a spanning tree using the following t steps procedure. First, let us denote by $\hat{V}$ the set of *supernodes*, which initially are the n singletons $\{\{u\} | u \in V\}$. At step $0 \leq i < t$, each node samples an incident edge to each set $\hat{v} \in \hat{V}$ using the ith collection of linear sketches $\sum_{x\in\hat{v}} S_{r_i}(a^x)$, and merge the obtained connected components into a single supernode. The procedure finishes before $t = \lceil \log n \rceil$ steps since the number of supernodes at least halves at each step. This idea is behind the proof of the following proposition.

Proposition 3 (Ahn et al. [1]). *Let $n, \delta > 0$ and $t = \lceil \log n \rceil$. There exists an algorithm that receives t independent δ-connectivity sketches of a graph G, produced with $r_1, \dots, r_t \in \{0,1\}^{\mathcal{O}(\log n)}$ random strings picked uniformly at random, and outputs a spanning forest of G with probability $1 - \delta$.*

Lemma 2. *There exists a one-round algorithm in the* BClique$[\lfloor k/2 \rfloor + 1]$ *model which computes, for every node $x \in V$, the connected components of $G - N^{\lfloor k/2 \rfloor}[x]$, using bandwidth $\mathcal{O}(\log^4 n)$ and with high probability.*

Proof. The algorithm works as follows. First, each node x sends $t = \lceil \log n \rceil$ different $1/n^2$-linear sketches of its connectivity vector a^x, using t random strings $r_1, \dots, r_t$. Note that each node knows F_x. Observe that the components of $G - N^{\lfloor k/2 \rfloor}[x]$ are exactly the components of $G - F_x$ without considering the nodes in $N^{\lfloor k/2 \rfloor}[x]$. In the following, we show that after the communication round, each node x can compute a spanning forest of $G - F_x$ with probability at least $1 - 1/n^2$. Therefore, the whole algorithm succeeds with probability at least $1 - 1/n$.

Let $S_r(G) = (S_r(a^{x_1}), \dots, S_r(a^{x_n}))$ be one of the $1/n^2$-connectivity sketches of G, produced with the random string r, received in the communication round. Consider, for each $e \in F_x$ and $u \in e$, the vector $b^{u,e}$ of dimension $\binom{n}{2}$ where,

$$b^{u,e}_{e'} = \begin{cases} -a^u_e & \text{if } e' = e, \\ 0 & \text{otherwise} \end{cases}, \text{ for each } e' \in \binom{n}{2}.$$

Let us call c^u be the connectivity vector of node u in $G - F_x$. Note that, for each $e \in \binom{n}{2}$,

$$c^u_e = a^u_e + \sum_{\{e' \in F_x : u \in e'\}} b^{u,e'}_e = \begin{cases} a^u_e \text{ if } e \in E(G) \setminus F_x, \\ 0 \text{ otherwise.} \end{cases}$$

If we define $S^u_r = S_r(a^u) + \sum_{\{e \in F_x : u \in e\}} S_r(b^{u,e})$, we obtain, by linearity of S_r, that $S^u_r = S_r(c^u)$ and then $\{S_r(c^u)\}_{u \in V}$ is a $1/n^2$-connectivity sketch of $G - F_x$ produced with r.

Then, after the communication round, any node x can obtain t different $1/n^2$-connectivity sketches of $G - F_x$ produced with random strings $r_1, \dots, r_t$ picked uniformly at random. Therefore, by Proposition 3, it can produce a spanning forest of that graph with probability at least $1 - 1/n^2$. □

4.2 Deciding *k*-Chordality

We are now able to express the distributed algorithm recognizing k-chordal graphs, see Algorithm 1.

Theorem 4. *Let $k \geq 3$. There exists a two-round randomized algorithm in the* BClique$[\lfloor k/2 \rfloor + 1]$ *model, that recognizes k-chordal graphs, and thus solves problem* Cycle$_{>k}$*, with bandwidth $\mathcal{O}(\log^4 n)$ and high probability.*

Proof. In the first round, each node $x \in G$ computes the connected components of $G - N^{\lfloor k/2 \rfloor}[x]$ using the algorithm of Lemma 2. After the first round, each node x uses its knowledge of G to locally reconstruct H_x^k by identifying the connected components $D_1, \ldots, D_p$ of $G - N^{\lfloor k/2 \rfloor}[x]$ and contracting each D_i into a unique vertex d_i. Note that x sees the edges between D_i and $N^{\lfloor k/2 \rfloor}[x]$. Finally, x checks whether H_x^k is k-chordal and communicates the answer in the second round. By Lemma 1, the input graph is chordal if and only if each vertex x communicated a YES answer. We emphasis that the second round is needed only because the nodes must all agree on the output.

The algorithm may fail only when some node x fails to compute the components of $G - N^{\lfloor k/2 \rfloor}[x]$; this event may occur, from Lemma 2, with probability at most $1/n$. □

Algorithm 1. k-chordality

```
1 Round 1
2     Run the algorithm of Lemma 2 to compute the components of
      $G - N^{\lfloor k/2 \rfloor}[x]$;
3 Round 2
4     Each node $x$ builds $H_x^k$ contracting each component of $G - N^{\lfloor k/2 \rfloor}[x]$ into a
      single node;
5     Each node $x$ checks whether $H_x^k$ is $k$-chordal and communicates the answer
      to the other nodes;
6     The graph is $k$-chordal if all messages communicated during this round are
      YES messages;
```

We end this section giving a lower-bound on the bandwidth b for any one-round algorithm solving Cycle$_{>k}$ in the BClique$_r$ model, when $0 < r \leq k/3$.

Theorem 5. *Let $\epsilon \leq 1/3$, and $0 < r \leq k/3$. Any ϵ-error, one-round algorithm in the* BClique$[r]$ *model that solves* Cycle$_{>k}$ *requires bandwidth $\Omega(n/\log n)$.*

5 Conclusion

All throughout the paper we considered problems Cycle$_{\leq k}$ and Cycle$_{>k}$. Let us briefly discuss the similar problems Sub-Cycle$_{\leq k}$, Sub-Cycle$_{>k}$ and Sub-Cycle$_{=k}$, which consist in deciding whether the input graphs has, as a subgraph, a cycle of length at most k, greater than k, and equal to k, respectively.

Observe that Sub-Cycle$_{\leq k}$ is identical to Cycle$_{\leq k}$, so upper and lower bounds coincide. We emphasize that, for $k \geq 3r$, the lower and upper bounds for these problems are tight up to polylogarithmic factors.

Unlike the case of short cycles, there is a significative difference between detecting long induced cycles and detecting long cycles (induced or not). By a result of Birmelé [9], graphs with no cycles of length greater than k have treewidth (and hence degeneracy) at most k. Therefore, they can be recognized by a one-round deterministic algorithm in the BClique model with bandwidth $\mathcal{O}(k \log n)$, based on Theorem 1.

Further lower bounds can be obtained for both $\text{Cycle}_{=k}$ and $\text{Sub-Cycle}_{=k}$ problems in the BClique[r] model, when k is an odd number between $3r$ and $4r$. These bounds are obtained by a reduction from a 3-party Number-On-the-Forehead version of the disjointness problem DISJ, and show that any deterministic R-round b-bandwidth algorithm for this problem, in the BClique[r] model, is such that $R \cdot b = \Omega(n^{1-o(1)})$. Under some stronger complexity assumptions, this lower bound can be extended to randomized algorithms.

When k is even, problem $\text{Sub-Cycle}_{=k}$ can be solved by a one-round deterministic algorithm in BClique with bandwidth $\mathcal{O}(n^{2/k} \log n)$, thanks to degeneracy arguments.

We leave as open problems the question whether $\text{Cycle}_{>k}$ can be solved by a non-trivial one-round algorithm in the BClique[$\lfloor k/2 \rfloor + 1$] model, as well as the question of multi-round lower bounds for this problem in the BClique[r] model for $r < k/2$.

References

1. Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 459–467 (2012)
2. Ahn, K.J., Guha, S., McGregor, A.: Graph sketches: sparsification, spanners, and subgraphs. In: Proceedings of the 31st Symposium on Principles of Database Systems, PODS 2012, pp. 5–14 (2012)
3. Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, pp. 82–93. ACM (1980)
4. Arfaoui, H., Fraigniaud, P., Ilcinkas, D., Mathieu, F.: Distributedly testing cycle-freeness. In: Kratsch, D., Todinca, I. (eds.) WG 2014. LNCS, vol. 8747, pp. 15–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12340-0_2
5. Awerbuch, B., Goldreich, O., Vainish, R., Peleg, D.: A trade-off between information and communication in broadcast protocols. J. ACM **37**(2), 238–256 (1990)
6. Becker, F., Kosowski, A., Matamala, M., Nisse, N., Rapaport, I., Suchan, K., Todinca, I.: Allowing each node to communicate only once in a distributed system: shared whiteboard models. Distrib. Comput. **28**(3), 189–200 (2015)
7. Becker, F., Matamala, M., Nisse, N., Rapaport, I., Suchan, K., Todinca, I.: Adding a referee to an interconnection network: what can(not) be computed in one round. In: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2011, pp. 508–514 (2011)
8. Becker, F., Montealegre, P., Rapaport, I., Todinca, I.: The simultaneous number-in-hand communication model for networks: private coins, public coins and determinism. In: Halldórsson, M.M. (ed.) SIROCCO 2014. LNCS, vol. 8576, pp. 83–95. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09620-9_8

9. Birmelé, E.: Tree-width and circumference of graphs. J. Graph Theory **43**(1), 24–25 (2003)
10. Bondy, J.A., Simonovits, M.: Cycles of even length in graphs. J. Comb. Theory, Ser. B **16**(2), 97–105 (1974)
11. Brandstädt, A., Spinrad, J.P., et al.: Graph classes: a survey, vol. 3. Siam (1999)
12. Censor-Hillel, K., Kaski, P., Korhonen, J.H., Lenzen, C., Paz, A., Suomela, J.: Algebraic methods in the congested clique. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, pp. 143–152 (2015)
13. Chandrasekharan, N., Sitharama Iyengar, S.: NC algorithms for recognizing chordal graphs and k trees. IEEE Trans. Comput. **37**(10), 1178–1183 (1988)
14. Drucker, A., Kuhn, F., Oshman, R.: On the power of the congested clique model. In: ACM Symposium on Principles of Distributed Computing, PODC 2014, pp. 367–376 (2014)
15. Erdős, P.: Extremal problems in graph theory. In: Theory of Graphs and its Applications. Proceedings of the Symposium, Smolenice (1964)
16. Guha, S., McGregor, A., Tench, D.: Vertex and hyperedge connectivity in dynamic graph streams. In: Proceedings of the 34th ACM Symposium on Principles of Database Systems, pp. 241–247. ACM (2015)
17. Hegeman, J.W., Pemmaraju, S.V., Sardeshmukh, V.B.: Near-constant-time distributed algorithms on a congested clique. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 514–530. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45174-8_35
18. Holzer, S., Pinsker, N.: Approximation of distances and shortest paths in the broadcast congest clique. In: 19th International Conference on Principles of Distributed Systems, OPODIS 2015, Leibniz International Proceedings in Informatics (LIPIcs), vol. 46, pp. 1–16 (2016)
19. Jowhari, H., Saglam, M., Tardos, G.: Tight bounds for lp samplers, finding duplicates in streams, and related problems. In: Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, pp. 49–58 (2011)
20. Jurdzinski, T., Nowicki, K.: MST in O(1) rounds of the congested clique. Preprint arXiv:1707.08484 (2017)
21. Kari, J., Matamala, M., Rapaport, I., Salo, V.: Solving the INDUCED SUBGRAPH problem in the randomized multiparty simultaneous messages model. In: Scheideler, C. (ed.) Structural Information and Communication Complexity. LNCS, vol. 9439, pp. 370–384. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25258-2_26
22. Lazebnik, F., Ustimenko, V.A., Woldar, A.J.: A new series of dense graphs of high girth. Bull. Am. Math. Soc. **32**(1), 73–79 (1995)
23. Linial, N.: Locality in distributed graph algorithms. SIAM J. Com. **21**(1), 193–201 (1992)
24. Montealegre, P., Todinca, I.: Brief anouncement: deterministic graph connectivity in the broadcast congested clique. In: Proceedings of the 35th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2016 (2016)
25. Naor, M., Stockmeyer, L.: What can be computed locally? SIAM J. Comput. **24**(6), 1259–1277 (1995)

Partitioning Orthogonal Histograms into Rectangular Boxes

Therese Biedl[1], Martin Derka[2], Veronika Irvine[1], Anna Lubiw[1], Debajyoti Mondal[3](✉), and Alexi Turcotte[1]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
[2] School of Computer Science, Carleton University, Ottawa, Canada
[3] Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
dmondal@cs.usask.ca

Abstract. The problem of partitioning an orthogonal polyhedron into a minimum number of boxes was shown to be NP-hard in 1991, but no approximability result is known except for a 4-approximation algorithm for 3D-histograms. In this paper we broaden the understanding of the 3D-histogram partitioning problem. We prove that partitioning a 3D-histogram into a minimum number of boxes is NP-hard, even for histograms of height two. This settles an open question posed by Floderus et al. We then show the problem to be APX-hard for histograms of height four. On the positive side, we give polynomial-time algorithms to compute optimal or approximate box partitions for some restricted but interesting classes of polyhedra and 3D-histograms.

1 Introduction

Partitioning a geometric object or a shape into simpler parts is a classic problem in computational geometry. Such partitioning problems are motivated by their applications in image processing, camera placement in security systems, computer graphics, VLSI manufacturing, and so on.

An important special case is when the object is an *orthogonal* polygon or polyhedron—meaning that the edges or faces are parallel to the axes or coordinate planes—and the goal is to partition into a minimum number of rectangles or *boxes*, where a *box* is an orthogonal polyhedron with 6 faces (i.e., the 3D equivalent of a rectangle).

In two dimensions the problem of partitioning an orthogonal polygon into a minimum number of rectangles can be solved in polynomial time, both for simple polygons and for polygons with holes [5,7,12–14]. In three dimensions the problem becomes NP-hard in general, as proved by Dielissen and Kaldewaij [3]. Our aim is to explore the boundary between hard and easy for a special class of orthogonal polyhedra, namely *histograms*.

A *2D-histogram* is an orthogonal polygon L that contains an edge e such that for any point $p \in L$, the line segment connecting p to its orthogonal projection on

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 146–160, 2018.
https://doi.org/10.1007/978-3-319-77404-6_12

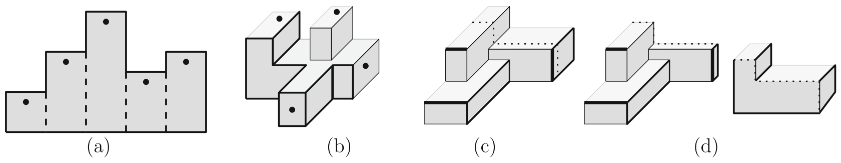


Fig. 1. (a) A 2D-histogram with a minimum partition into rectangles, as witnessed by the given "independent" points, no two of which lie in a rectangle. (b) A 3D-histogram requiring 4 boxes in its optimal partition since there are four "independent" points (no two in a box). (c)–(d) A guillotine cut.

e lies entirely inside L. See Fig. 1(a). Similarly, a *3D-histogram* is an orthogonal polyhedron H that contains a face f such that for any point $p \in H$, the line segment connecting p to its orthogonal projection on f lies entirely inside H. See Fig. 1(b). The face f is called the *base* of the histogram. Note that any histogram can be adjusted so that the vertices (and consequently also edges and faces) have integer coordinates, and the combinatorial structure of the histogram is preserved. Throughout this paper, we assume that any histogram has integer coordinates and that the base has z-coordinate 0 and all other faces lie above the base. The *height* of a face parallel to the base is its z-coordinate, and the maximum of these is called the *height* of the histogram.

Floderus et al. [8] gave an $O(n \log n)$-time 4-approximation algorithm to partition a 3D-histogram into a minimum number of boxes and asked whether the problem is NP-hard for histograms. We note that the NP-hardness reduction of Dielissen and Kaldewaij [3] does not hold for histograms.

Contributions: In this paper, we prove that partitioning 3D-histograms (even with height 2) into a minimum number of boxes is NP-hard (Sect. 3). The problem is APX-hard for 3D-histograms of height 4 (Sect. 4). We show that optimal partitioning must consider cuts beyond those that are "guillotine" (Sect. 5). We then focus on restricted classes of polyhedron (Sects. 6–7). If two dimensions of the polyhedron are fixed, then we compute a minimum box partition in polynomial time. If one dimension is bounded by t, then we produce a t-approximation in polynomial time. Finally, we give a polynomial-time 2-approximation algorithm for the box partition of corner polyhedra.

Background: The problem of partitioning an orthogonal polygon into rectangles has also been considered with different objective functions, for example, minimizing the total length of the cuts ("minimum ink") [11], avoiding very thin rectangles by minimizing the aspect ratio [16], or minimizing the so-called "stabbing number" [4]. Computing a minimum decomposition of arbitrary polygons with holes into (perhaps overlapping) convex, star-shaped, or spiral subsets is NP-hard [15]. If Steiner points are not permitted, then some partition and covering problems become polynomial-time solvable for arbitrary simple polygons [10].

2 Preliminaries

The *top surface* of a 3D-histogram H denotes the union of the faces parallel to the base, excluding the base itself. For any integer $h \geq 1$, an *h-region* of the histogram is a maximal region that all has the same face f as top surface, and f has height h.

For a point p in $\mathbb{R}^3$, we denote its x, y and z-coordinates by p_x, p_y and p_z, respectively. A set of points in H is called *independent* if there does not exist any box in H that contains two or more of these points. A set of edges in H is called a set of *forcing edges* if there does not exist any box in H that properly intersects two or more of these edges. This implies that the midpoints of the forcing edges form an independent set of points. Consequently, if there are k forcing edges in a 3D-histogram, then k is a lower bound on the size of any box partition of the histogram.

Let L be a plane parallel to one of the axis planes. We say that a *guillotine cut* along L partitions a polyhedron H into two polyhedra H_1 and H_2 if $H_1 \cup H_2 = H$ and $H_1 \cap H_2 \subset L$ (Fig. 1(c)–(d)).

Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges. We call G a *planar graph* if it admits a drawing on the Euclidean plane such that no two edges cross except possibly at a common end-point. G is *cubic* if the degree of every vertex in G is exactly three. A *vertex cover* of G is a set of vertices C in G such that for every edge (v, w), at least one of v and w belongs to C.

3 3D-Histogram Partition is NP-Hard for Height $\geq$2

In this section we prove that partitioning a 3D-histogram into a minimum number of boxes (*3D-Histogram Partition*) is NP-hard even when the histogram has height two. We reduce from the problem of computing a minimum-cardinality vertex cover in a cubic planar graph, which is NP-hard [18].

Let G be a cubic planar graph and let G' be the *2-subdivision of* G, defined to be the graph obtained from G by replacing each edge of G by a path with three edges, of which the middle one is a double edge. Observe that G' is also cubic and planar. The crucial idea for constructing a histogram is to use a suitable drawing of G'. Here, an *orthogonal drawing* of a planar graph G is a planar drawing of G such that each vertex is mapped to a point in the Euclidean plane, and each edge is mapped to an axis-aligned polyline between the corresponding points. It is called *1-bend* if every polyline has exactly one bend.

Lemma 1. *Let G be a cubic planar graph. Then the 2-subdivision G' of G admits a 1-bend orthogonal drawing Γ'.*

Proof. (Sketch) Take an orthogonal drawing Γ of G with at most two bends per edge, which can be constructed in linear time [9]. For any edge without bends in Γ, apply a so-called zig-zag transformation so that it obtains exactly two bends. We obtain Γ' by replacing the drawing of each edge of e with the drawing of a path with a double edge along e's poly-line, see Fig. 2(b)–(c). $\square$

Construction of H: Graph G' has $n' = n + 2m$ vertices and $m' = \frac{3}{2}n'$ edges. We construct a histogram H that can be partitioned into $(4n'+3m'+\alpha')$ boxes if and only if G' contains a vertex cover of size α'. Note that G' contains a vertex cover of size α' if and only if G contains a vertex cover of size $\alpha' - m$ (see e.g., see [17]), so this then proves the reduction.

We transform the 1-bend drawing Γ' of G' into the desired histogram H. Specifically, we replace each vertex v of Γ' (i.e., both original and subdivision vertices) by a vertex gadget $\lambda(v)$ (see Fig. 2(d)–(e)). The numbers in Fig. 2(d) illustrate the heights of the corresponding regions. We then replace each edge (v, w) of Γ' using an edge gadget $\lambda(v, w)$ (see Fig. 2(f)). The edge gadgets corresponding to the edges incident to v are attached to the three sides of height 1 of $\lambda(v)$. This completes the construction of H.

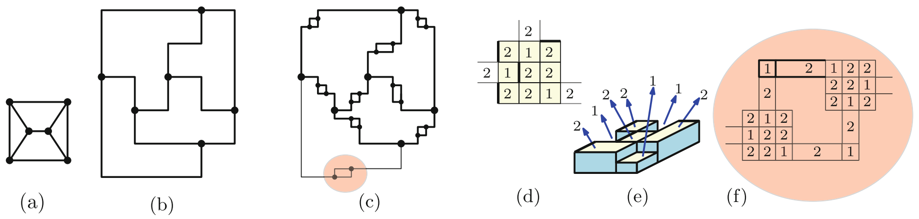


Fig. 2. (a) A cubic graph G. (b) Its orthogonal drawing Γ. (c) A 1-bend orthogonal drawing Γ' of G'. (d) Top view of a vertex gadget, where the edge connections are shown in gray. (e) Side view of a vertex gadget. (f) Connecting vertex gadgets using edge gadgets.

Let Opt(H) be a partition of H into a minimum number of boxes. The following lemmas (whose proofs are omitted) discuss some properties of the gadgets with respect to Opt(H). In brief, Lemma 2 follows from the forcing edges that are illustrated in Fig. 2(c). Lemmas 3–4 follow from the observation that a box that touches a face of height 2 (on the top surface) cannot touch a face of height 1, and that the edge gadgets are "non-aligned".

Lemma 2. *For every vertex gadget $\lambda(v)$,* Opt(H) *contains at least four distinct boxes that lie entirely inside $\lambda(v)$. If it contains exactly four such boxes, then none of the 1-regions in $\lambda(v)$ are covered by these boxes.*

Lemma 3. *For every edge gadget $\lambda(v, w)$,* Opt(H) *must contain 3 boxes that intersect $\lambda(v, w)$. No box in* Opt(H) *can intersect more than one edge gadget.*

Lemma 4. *If an edge gadget $\lambda(v, w)$ is entirely covered by exactly three boxes in* Opt(H)*, then these three boxes cover at most one of the two 1-regions of $\lambda(v)$ and $\lambda(w)$ that are adjacent to $\lambda(v, w)$ (e.g., see Fig. 3(a)–(d)).*

Equivalence Between Instances: Given a set of r boxes, it is straightforward to verify whether the boxes are interior disjoint and cover the input histogram

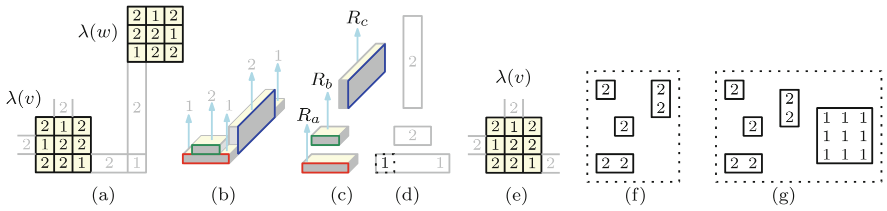


Fig. 3. (a)–(b) Illustration for $\lambda(v, w)$. (c) A partition of the edge gadget that covers the 1-region of v at (v, w). (d) A schematic representation of the partition. (e)–(g) Illustration for Lemma 5.

in polynomial time. Hence the problem 3D-Histogram Partition is in NP. Since H can be constructed in polynomial time, we can use the following lemma to obtain the NP-hardness.

Lemma 5. *G' contains a vertex cover $\mathcal{C}$ of size α' if and only if H can be partitioned into $(4n' + 3m' + \alpha')$ boxes.*

Proof (sketch). We construct a partition of H from a vertex cover, as follows.

A. If $v \notin \mathcal{C}$, then we use four maximal boxes to cover the 2-regions of $\lambda(v)$, as illustrated in Fig. 3(e)–(f). The remaining regions of $\lambda(v)$ are 1-regions, which will be covered by the boxes partitioning the edge gadgets.
B. If $v \in \mathcal{C}$, then we use 5 maximal boxes to cover $\lambda(v)$, e.g., see Fig. 3(g).
C. We use three maximal boxes to cover each edge gadget $\lambda(v, w)$. Note that either v or w must lie in $\mathcal{C}$. If $v \notin \mathcal{C}$, then one of these boxes will cover the 1-region of $\lambda(v)$ at $\lambda(v, w)$, e.g., see Fig. 3(a)–(d). Similarly, if $w \notin \mathcal{C}$, then one of these boxes will cover the 1-region of $\lambda(w)$ at $\lambda(v, w)$.

One easily verifies that Steps A–C partition the histogram H into $(3m' + 4n' + \alpha')$ boxes. For the other direction, assume that H admits a partition $\mathcal{B}$ with $(3m' + 4n' + \alpha')$ boxes, and construct a vertex cover of size at most α' in G'. By Lemma 2, every vertex gadget contributes to at least four distinct boxes in $\mathcal{B}$, which corresponds to the 2-regions. Hence we use at least $4n'$ boxes of $\mathcal{B}$ to cover those regions. Note that all these boxes lie entirely inside the vertex gadgets. By Lemma 3, every edge gadget must use at least three distinct boxes, which altogether sum up to at least $3m'$. These boxes may also cover some 1-regions of the vertex gadgets (e.g., see Lemma 4). Since $\mathcal{B}$ contains at most $(3m' + 4n' + \alpha')$ boxes, we have at most α' boxes remaining to cover the remaining 1-regions of the vertex gadgets. We now construct a set $\mathcal{S}$ as follows: (a) If a vertex gadget $\lambda(v)$ contains more than four boxes lying entirely inside $\lambda(v)$, then we include v into $\mathcal{S}$. (b) If four or more boxes intersect an edge gadget $\lambda(v, w)$, then we choose one of v and w arbitrarily into $\mathcal{S}$.

Note that each step can be charged uniquely to one of the remaining α' boxes, i.e., the box charged in Step (a) lies entirely inside $\lambda(v)$ and hence cannot be charged again in Step (b). Hence the number of vertices in $\mathcal{S}$ is at most α'. One argues that $\mathcal{S}$ is a vertex cover based on the observation that either (a) or (b) must apply for each edge gadget. □

Theorem 1. *Partitioning a 3D-histogram into a minimum number of boxes is NP-hard, even when the histogram is of height two.*

4 3D-Histogram Partition is APX-Hard for Height ≥4

In this section we prove that partitioning a 3D-histogram into a minimum number of boxes is APX-hard, even for histograms of height 4. We reduce from the problem of computing a minimum-cardinality vertex cover in a cubic graph (not necessarily planar), which is APX-hard [1].

Construction of H: Let $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$ be the vertices and edges of G. Consider an integer grid of size $(8n+1) \times (2n+10m+1)$. Column $8i-3$ is assigned to v_i, and column $8i+1$ is assigned to the transition from v_i to v_{i+1} (we write $w_{i,i+1}$ for short as in Fig. 4(b)). Below the grid we add a staircase that descends at each column of v_i or $w_{i,i+1}$, and here add a *tooth*, i.e., a square for which all but the top sides are on the boundary. Edge e_j is assigned to row $10i+2n+4$. For each edge, cut out H-shaped holes in the polygon where the row of the edge meets the columns of its endpoints. These holes have width 7, height 7 or more, and remove four (five) squares from the column of the left (right) endpoint. The resulting polygon P (Fig. 4(a)) forms the base of the histogram (except for some additions via edge gadgets that will be listed below). Extrude all of P to height 2; we call the result the *platform*.

For each edge $e = (v, w)$, we add an *edge gadget* $\lambda(v, w)$ that consists of two *endpoint gadgets* and a *connector*. Here, the endpoint gadget at v consists of the four (five) squares from the vertex column of v (we call these the *decision column* $\lambda(v, e)$) as well as a surrounding polygon; all of these have height 1 (see Fig. 4(d)–(e)). The *connector* connects the two endpoint gadgets via a sequence of 2-regions, 3-regions and 4-regions along the row of the edge; it sits partially on the endpoint gadgets and partially on the platform (see Fig. 4(f)–(g)). This completes the construction of H.

To argue the correctness, we fix a set F of $13n + 1$ edges (shown in bold in Fig. 4(b)) that can easily be seen to be forcing edges. We use the $2n + 1$ horizontal top edges of the teeth, as well as the $2n$ horizontal top edges that lie between these teeth. For each edge, we select 6 further horizontal edges from the hole boundaries of its two endpoints, see Fig. 4(b); we assume that these hole boundaries were chosen that none of them have the same y-coordinate. Consequently, we obtain a set of $(2n + 1) + (2n) + (6m) = 4n + 9n + 1$ forcing edges, all of which are on top of the platform.

We must argue how many boxes are needed to cover most (but not all) of an edge gadget $\lambda(v, w)$. We say that a *sub-partition of* $\lambda(v, w)$ is a set of disjoint boxes that cover the entire edge gadget except that they may leave one or both decision columns $\lambda(v, e)$ or $\lambda(w, e)$ uncovered. Given a partition of the histogram, we *charge* a box B to edge-gadget $\lambda(v, w)$ if either B intersects the interior of $\lambda(v, w)$ or if B lies inside the platform and the top front edge of B lies on the

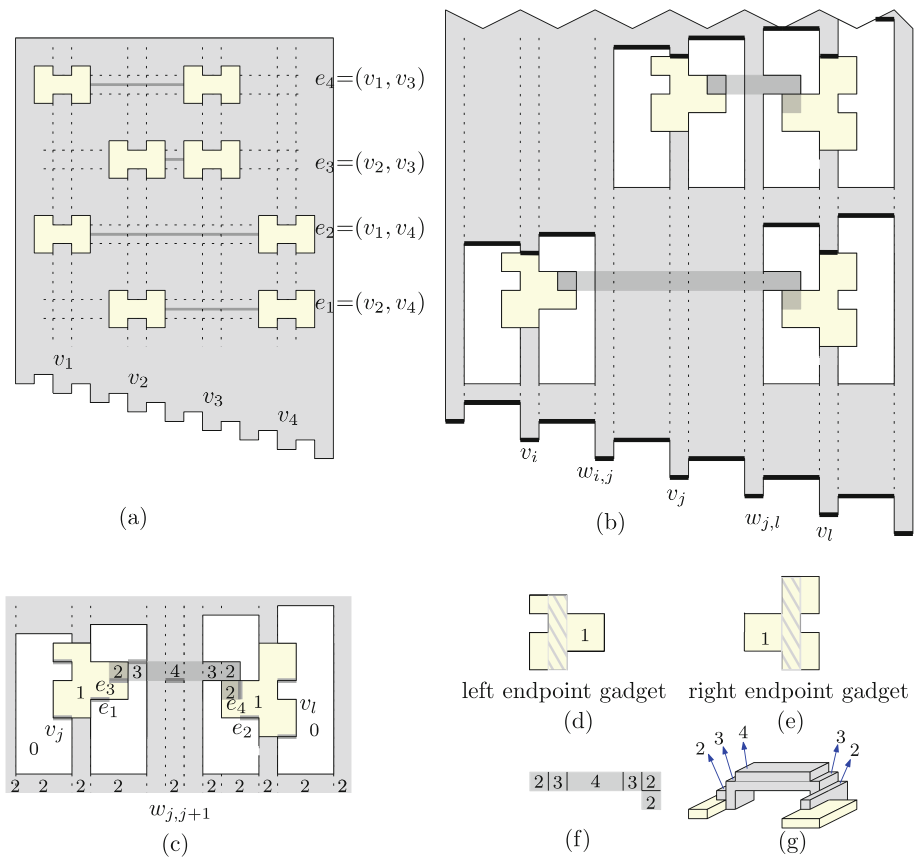


Fig. 4. (a) Schematic representation and (b) top view of H. (c) Illustration for the heights near one edge gadget. (d)–(e) Endpoint gadgets and decision columns. (f)–(g) A connector gadget is shown in gray.

boundary of the 4-region of $\lambda(v, w)$. (In particular, such a box cannot cover an edge in F and it can only be charged to one edge-gadget.) Crucial for the reduction is the following lemma:

Lemma 6. *The edge gadgets satisfy the following properties: (P_1) Every sub-partition of an edge gadget has least 12 boxes charged to it. (P_2) Every partition of an edge gadget (covering both decision columns) has at least 13 boxes charged to it.*

Proof (sketch). Consider a (sub-)partition $\mathcal{B}$ of some edge gadget $\lambda(v, w)$. Figure 4(c) shows some forcing edges of $\lambda(v, w)$. Of these, there are three each in the endpoint gadgets, forcing three boxes each, and four more in the connector-gadget.

In fact, the connector-gadget requires five boxes to be charged, as can be seen as follows: If no box of $\mathcal{B}$ intersects the platform below the connector-gadget,

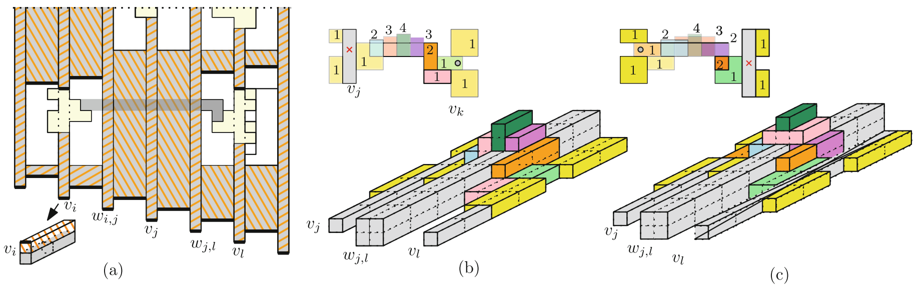


Fig. 5. (a) The maximal boxes determined by the forcing edges. (b)–(c) Partition of the endpoint gadget excluding the left and right decision column, respectively.

then we can find another independent point (on the downward-facing face of the 3-region on the right in Fig. 4(c), at height 1.5). If some box B of $\mathcal{B}$ intersects the platform, then some other box of the partition of $\mathcal{P}$ must share a side with B, and this other box is also charged to $\lambda(v, w)$.

Thus we have now 11 boxes to be charged to $\lambda(v, w)$. One can also easily verify that if at one connector-gadget the decision column is covered, then this requires one additional box not counted elsewhere. This proves the claim in all cases except the one where neither decision-column is covered, each endpoint-gadget uses exactly three boxes, and the connector-gadget has exactly five boxes charged to it. One can verify that this is impossible if none of the boxes overlap. □

Equivalence Between Instances: We now prove that H can be partitioned into $(|F| + 12m + k)$ boxes if and only if G has a vertex cover of size at most k.

Lemma 7. *If G has a vertex cover $\mathcal{C}$ of size k, then H admits a partition into $(|F| + 12m + k)$ boxes.*

Proof (sketch). For each vertex $v \in \mathcal{C}$, construct a box that has width and height 1 and whose depth is so large that is spans the entire column of v. In particular it covers all decision columns $\lambda(v, e)$ of incident edges of v, e.g., see the gray box for v_i in Fig. 5(a). The rest of the platform can be covered using one box per forcing edge. Finally, there are sub-partitions of an edge gadget $\lambda(v, w)$ using 12 boxes (e.g., Fig. 5(b)–(c)) so that $\lambda(v, e)$, $\lambda(w, e)$, or neither, is covered. Applying the suitable one to each edge (depending on whether $v \in \mathcal{C}$, $w \in \mathcal{C}$, or $v, w \in \mathcal{C}$) gives the desired partition. □

Lemma 8. *If H admits a partition $\mathcal{B}$ into $(|F| + 12m + k)$ boxes, then G has a vertex cover of size at most k.*

Proof. We construct a vertex cover $\mathcal{C}$ of G as follows: (a) For every decision column $\lambda(v, e)$, if the box covering it lies entirely in the column of v, then add v to $\mathcal{C}$. (b) For an edge (v, w), if neither v nor w belongs to $\mathcal{C}$, and at least 13 boxes are charged to $\lambda(v, w)$, then arbitrarily add one of v and w to $\mathcal{C}$.

We first show that $\mathcal{C}$ contains at most k vertices. The set F of forcing edges determines $|F|$ boxes that we denote by R_F. All of them cover no decision column and cover no part of an edge-gadget or are charged to it. $\mathcal{B} - R_F$ has $12m + k$ boxes. By Lemma 6 at least 12 of them are charged to each edge. This leaves at most k boxes that lead to an addition to $\mathcal{C}$.

Suppose now for a contradiction that for some edge $e = (v, w)$ neither v nor w belongs to $\mathcal{C}$. Then the boxes covering $\lambda(v, e)$ and $\lambda(w, e)$ cannot lie entirely inside their corresponding vertex columns. In other words, Step (a) did not apply. In this scenario, both these decision columns are covered using boxes from the edge-gadget, so by Lemma 6 at least 13 boxes are charged to $\lambda(v, w)$. Hence by Step (b), either v or w must belong to $\mathcal{C}$. □

Theorem 2. 3D-HISTOGRAM-PARTITION *is APX-hard.*

Proof. Let $\mathcal{C}^*$ and S^* be the optimum vertex cover of G and the optimum box partition of H, respectively. Assume that we had an $(1 + \epsilon)$-approximation algorithm for 3D-HISTOGRAM-PARTITION that computes a solution S from which we extract a vertex cover $\mathcal{C}$. By Lemmas 7–8, we have $|S| = |F| + 12m + |\mathcal{C}|$ and $|S^*| = |F| + 12m + |\mathcal{C}^*|$. Since G is cubic, $|C^*| \geq n/3$. Finally observe that $|S^*| \leq |F| + 12m + |C^*| \leq 13n + 1 + 18n + (n - 1) = 32n$. Hence we get

$$\frac{|\mathcal{C}|}{|\mathcal{C}^*|} = \frac{|S| - |F| - 12m}{|S^*| - |F| - 12m} \leq \frac{(1+\epsilon)|S^*| - |F| - 12m}{|S^*| - |F| - 12m} = 1 + \frac{\epsilon|S^*|}{|\mathcal{C}^*|} \leq 1 + \frac{32n\epsilon}{n/3} = 1 + 96\epsilon,$$

implying an approximation algorithm for vertex cover. The APX-hardness of 3D-HISTOGRAM-PARTITION now follows from the APX-hardness of minimum vertex cover. □

5 Partitioning Using Guillotine Cuts

In this section we show that there is an infinite family of 3D-histograms that cannot be optimally partitioned using guillotine cuts, whereas 2D-polygons can be partitioned optimally using such cuts by first cutting along "good diagonals" [5].

We say that $\mathcal{P}$ is a partitioning of a polyhedron H into boxes using guillotine cuts if $\mathcal{P}$ is a partition of H into boxes and there is a sequence $\mathcal{P}_0 = \{H\}, \mathcal{P}_1, \ldots, \mathcal{P}_k = \mathcal{P}$ of sets of polyhedra such that every $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by partitioning $\mathcal{P}_i$ using guillotine cuts.

Theorem 3. *There is an infinite family of 3D-histograms that cannot be partitioned optimally using only guillotine cuts.*

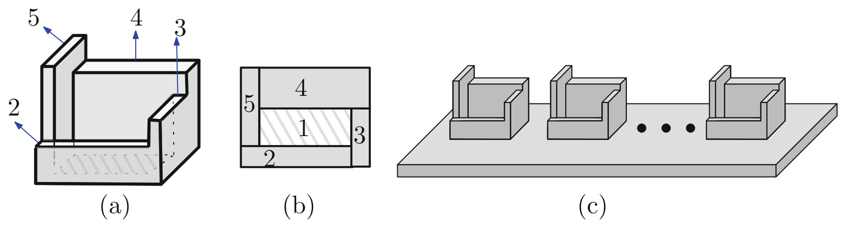


Fig. 6. (a) Illustration for H. (b) The top view of H. (c) Construction of H_k.

Proof. We first refer the reader to the histogram H of Fig. 6(a). Since H has five faces of distinct height, any partition of H into boxes would require 5 boxes, and H admits such a partition (e.g. by cutting along the edges of the top view in Fig. 6(b)). We now show that H cannot be partitioned into 5 boxes if we restrict the cuts to be guillotine.

Consider starting with a vertical guillotine cut, i.e., a cut perpendicular to the x-axis or y-axis. Any such cut results in two polyhedra: one with at least 4 faces of distinct height, and another with at least 2 faces of distinct height. Any further cutting of these polyhedra will result in at least 6 boxes, a contradiction.

If instead we start with a horizontal cut (perpendicular to the z-axis), we have 4 choices: cutting at heights 1, 2, 3, or 4. Cutting at any height other than 1 results in two polyhedra, one of which with 5 boxes in its optimal partitioning, a contradiction. Assume that we start by cutting at height 1. Any subsequent vertical guillotine cut results in two polyhedra: one with at least 3 faces of distinct height, and another with at least 2. Therefore, no vertical guillotine cuts are acceptable. Any subsequent horizontal cut immediately result in a contradiction: one of the polyhedra resulting from such a horizontal guillotine cut has 4 boxes in its optimal partitioning, and together with the 1 box from the first cut and the (at least) 1 other box from this cut we have at least 6 boxes.

It is now straightforward to create an infinite family of polyhedra $H_1 (= H), H_2, \ldots, H_k$ by attaching up to k copies of H on a rectangular box, as in Fig. 6(c), such that none of them can be partitioned optimally using only guillotine cuts. □

6 Orthogonal Polyhedra with Bounded Dimensions

In this section we focus on orthogonal polyhedra with bounded dimensions (recall that all vertex-coordinates are assumed to be integers). If one dimension of the input polyhedron $\mathcal{P}$ is bounded by t, then we construct a t-approximate box partition. If two dimensions are bounded, then we compute an optimal box partition in polynomial time. This works for all polyhedra of bounded dimensions, even if they are not histograms or have holes.

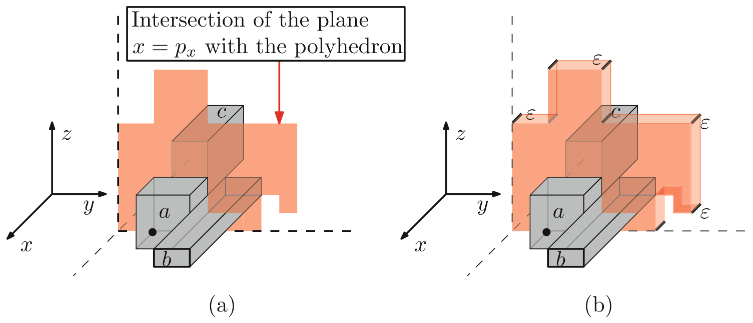


Fig. 7. Illustration for Lemma 10.

So let $\mathcal{P}$ be a polyhedron that resides within the $[0, W] \times [0, L] \times [0, H]$ box. If H is bounded by t, then we partition the polyhedron into (up to) t sets of polyhedra $\mathcal{P}_i$ where $0 \leq i < t$ and $\mathcal{P}_i$ is bounded by the planes $z = i, z = i + 1$. For each $\mathcal{P}_i$, we compute an optimal box partition $B_i^* = \textsc{Opt}(\mathcal{P}_i)$ using the algorithm for partitioning 2D-polygons [5]. We claim that $\bigcup_i B_i^*$ gives a partition that is within a factor of t of the optimum.

Fix an optimal partition $\textsc{Opt}(\mathcal{P})$ and partition it by the planes $z = i$. Let B_i be the boxes between the planes $z = i$ and $z = i + 1$. Then $|B_i| \geq |B_i^*|$ but also $|B_i| \leq |\textsc{Opt}(\mathcal{P})|$. Hence $\sum_i |B_i^*| \leq \sum_i |B_i| \leq t \cdot |\textsc{Opt}(\mathcal{P})|$ and we have:

Lemma 9. *For orthogonal polyhedra with one dimension bounded by t, a minimum box partition can be approximated within a factor of t in polynomial time.*

Consider now the scenario when two dimensions are fixed, e.g., $W \cdot L \in O(1)$. We rely on the following lemma.

Lemma 10. *Any orthogonal polyhedron $\mathcal{P}$ with vertices having integer coordinates can be optimally partitioned into boxes where the coordinates are integral.*

Proof (sketch). Assume that in a partition some box-face is within a plane (say plane $x = p_x$) for which p_x is not integral. Then we can shift all box-boundaries within that plane to lie within $x = p_x + \varepsilon$ instead (see Fig. 7) to get closer to a solution where all coordinates are integral. □

Let the *voxel* $v_{i,j,k}$ be the unit cube $[i-1, i] \times [j-1, j] \times [k-1, k]$ and let the *column* $c_{i,j}$ be all the voxels $\{v_{i,j,k} : 1 \leq k \leq H\}$. (In the following, whenever the range of i, j is unspecified then we mean $1 \leq i \leq W$ and $1 \leq j \leq L$.) We have $W \cdot L \in O(1)$ columns, and hence $O(H)$ voxels. Consider some box partition $\mathcal{B}$ of $\mathcal{P}$ that uses only boxes with integer coordinates. Let $\mathcal{B}'$ be some set of boxes obtained from $\mathcal{B}$ by removing (repeatedly) some box whose entire top is visible to infinity, or becomes visible after some other boxes in $\mathcal{B} - \mathcal{B}'$ were removed. Boxes $\mathcal{B}'$ cover a subset $\mathcal{P}'$ of $\mathcal{P}$, and this subset can be described as follows: For each column $c_{i,j}$, $\mathcal{P}'$ contains all voxels of $c_{i,j}$ that belong to $\mathcal{P}$, up to some limit $b_{i,j}$, and then contains no other voxels of $c_{i,j}$.

This observation is the key idea for a dynamic programming algorithm to find the optimum partition of $\mathcal{P}$. For any integer values $b_{i,j}$, define the polyhedron $\mathcal{P}[\{b_{i,j}\}_{i,j}]$ to be the polyhedron obtained from $\mathcal{P}$ by removing for each column $c_{i,j}$ all voxels $v_{i,j,q}$ with $q > b_{i,j}$. For each such polyhedron, we compute (recursively) the size $T[\{b_{i,j}\}_{i,j}]$ of the optimal box partition. This gives the optimal box partition for $\mathcal{P} = \mathcal{P}[\{H\}_{i,j}]$.

We can fill an entry $T[\{b_{i,j}\}_{i,j}]$ by considering any box $B = [i_1, i_2] \times [j_1, j_2] \times [k_1, k_2]$ that could be part of a box partition of $\mathcal{P}[\{b_{i,j}\}_{i,j}]$ such that the top face of B is visible to infinity. In particular, we must have $b_{i,j} = k_2$ and $v_{i,j,k} \subset \mathcal{P}$ for all $i_1 < i \leq i_2, j_1 < j \leq j_2$, and $k_1 < k \leq k_2$. If this is satisfied, then one possible value for $T[\{b_{i,j}\}_{i,j}]$ is to add one to the value for $T[\{b'_{i,j}\}_{i,j}]$ (where $b'_{i,j} = k_1$ for all $i_1 < i \leq i_2, j_1 < j \leq j_2$ and $b'_{i,j} = b_{i,j}$ otherwise).

There are $W^2L^2 \in O(1)$ possibilities for i_1, i_2, j_1, j_2, and for each of them, we can find the only possible value k_2 in $O(W \cdot L) = O(1)$ time and all possible values of k_1 in $O(H)$ time. One update to the table can hence be done in $O(H)$ time. There are $O(H^{WL})$ table-entries, so the entire dynamic program takes $O(H^{O(1)})$ time. We may assume that any plane $z = i$ for integral i contains at least one vertex of $\mathcal{P}$ (else we could shrink the polyhedron to obtain a combinatorially equivalent one) so that $H \in \Theta(n)$. Therefore we can find the optimal partition in $O(n^{O(1)})$ time.

Theorem 4. *Given an orthogonal polyhedron $\mathcal{P}$ such that two dimensions of $\mathcal{P}$ are bounded, one can compute a minimum box partition of $\mathcal{P}$ in polynomial time.*

7 Corner Polyhedra

In this section we give a polynomial-time algorithm with approximation factor 2 for partitioning a *corner polyhedron* into a minimum number of rectangular boxes. This improves on the approximation factor of 4 for histograms.

A *corner polyhedron* (as defined by Eppstein [6]) is an orthogonal polyhedron in which all but three "back" faces are oriented towards the vector $(1, 1, 1)$, i.e., visible from a point at infinity on the line $x = y = z$. See Fig. 8. Without loss of generality we will assume that the three back faces intersect at the vertex $(0, 0, 0)$. A corner polyhedron can be drawn in the plane by isometric projection with all vertices except $(0, 0, 0)$ visible. For any point $p = (p_x, p_y, p_z)$ inside a corner polyhedron, the three orthogonal line segments connecting p to the planes $z = 0$, $y = 0$, and $x = 0$ are contained in the polyhedron. This implies that a corner polyhedron is a histogram with any of the three back faces as the base.

A *peak* of a corner polyhedron is a vertex that is a local maximum in the direction $(1, 1, 1)$. Equivalently a peak is a vertex where the solid angle is $4\pi/8$, not including the vertices that lie on the axis planes. Let k be the number of peaks. Observe that the peaks form a set of independent points, thus k is a lower bound on the number of boxes needed in a partition. We will show that any corner polyhedron can be partitioned into $2k$ boxes, which gives the approximation factor 2.

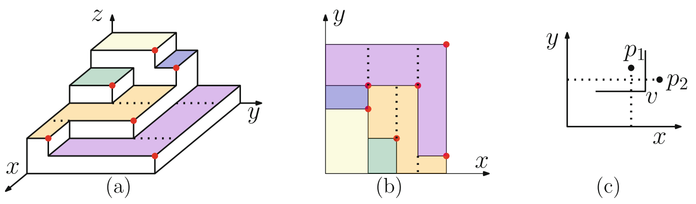


Fig. 8. (a) $\mathcal{P}$, with peaks shown as red dots. (b) $\mathcal{H}$, the projection of $\mathcal{P}$ to the $z = 0$ plane, and a partition of $\mathcal{H}$ into rectangles. (c) A forbidden corner in $\mathcal{H}$. (Color figure online)

Lemma 11. *A corner polyhedron $\mathcal{P}$ with k peaks can be partitioned into $2k$ boxes. Furthermore, such a partition can be found in polynomial time.*

Proof. Project $\mathcal{P}$ onto one of the axis planes, say $z = 0$. Call the result $\mathcal{H}$. Then $\mathcal{H}$ is a histogram partitioned into orthogonal polygons that correspond to the top faces of $\mathcal{P}$.

We claim that each such polygon consists of two *monotone* chains, each consisting of edges in the $+x$ and $-y$ directions. See Fig. 8(b). To justify this claim, observe that if a polygon of $\mathcal{H}$ were not monotone then it would have a vertex v where one edge goes to the left and one edge goes up, as shown in Fig. 8(c). Let p_1 be a point just above and left of v, and let p_2 be a point just above and right of v. Assume that p_1 and p_2 are the projections of points q_1 and q_2 on the surface of $\mathcal{P}$. We consider their z coordinates. If $z(q_1) > z(q_2)$ then a line from q_1 to the $y = 0$ plane leaves $\mathcal{P}$, and if $z(q_1) < z(q_2)$ then a line from q_2 to the $x = 0$ plane leaves $\mathcal{P}$—a contradiction in both cases.

Given that the polygons of $\mathcal{H}$ are monotone, we can partition them into rectangles by adding, from each peak vertex in $\mathcal{H}$, two vertical (i.e., in the y direction) segments, one going upwards and one going downwards. Each line segment stops when it is blocked by the interior of a horizontal edge of $\mathcal{H}$. This partitions $\mathcal{H}$ into at most $2k$ rectangles. Expand each rectangle into a 3D box from $z = 0$ to the maximum possible height. The result is $2k$ boxes that partition $\mathcal{P}$. See the final partition of $\mathcal{P}$ in Fig. 8(a). □

Theorem 5. *There is a polynomial-time 2-approximation algorithm to partition a corner polyhedron into boxes.*

8 Open Problems

1. Is there a constant-factor approximation algorithm for the case of general 3D orthogonal polyhedra?
2. For histograms, there is a 4-approximation algorithm [8]. Can the approximation factor of 4 be reduced?
3. For corner polyhedra, we gave a 2-approximation algorithm. Is there a PTAS, or even a polynomial-time algorithm?

4. What about other special cases of histograms, for example "convex polyhedra" [3] and "orthoballs" [2]?

For all these questions, the concept of independent points may be useful. While there are histograms for which the optimal box partition has a higher cardinality than any independent set of points (e.g. the one in Fig. 1(c) has only three independent points but requires four boxes), the maximum cardinality of an independent set of points could serve as a lower bound for an approximation algorithm. Can it be computed efficiently?

Acknowledgements. This work was done as part of the Algorithms Problem Session at the University of Waterloo. We thank the other participants for valuable discussions. Research of T.B. and A.L. supported by NSERC, M.D. supported by Vanier CGS, M.D. and D.M. supported by an NSERC PDF.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoret. Comput. Sci. **237**(1–2), 123–134 (2000)
2. Barrera-Cruz, F., Biedl, T.C., Derka, M., Kiazyk, S., Lubiw, A., Vosoughpour, H.: Turning orthogonally convex polyhedra into orthoballs. In: Proceedings of CCCG (2014)
3. Dielissen, V.J., Kaldewaij, A.: Rectangular partition is polynomial in two dimensions but NP-complete in three. Inf. Process. Lett. **38**(1), 1–6 (1991)
4. Durocher, S., Mehrabi, S.: Computing conforming partitions of orthogonal polygons with minimum stabbing number. Theor. Comput. Sci. **689**, 157–168 (2017)
5. Eppstein, D.: Graph-theoretic solutions to computational geometry problems. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 1–16. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11409-0_1
6. Eppstein, D., Mumford, E.: Steinitz theorems for simple orthogonal polyhedra. J. Comput. Geom. **5**(1), 179–244 (2014)
7. Ferrari, L., Sankar, P.V., Sklansky, J.: Minimal rectangular partitions of digitized blobs. Comput. Vis. Graph. Image Process. **28**(1), 58–71 (1984)
8. Floderus, P., Jansson, J., Levcopoulos, C., Lingas, A., Sledneu, D.: 3D rectangulations and geometric matrix multiplication. Algorithmica (2016, in press)
9. Kant, G.: Drawing planar graphs using the canonical ordering. Algorithmica **16**(1), 4–32 (1996)
10. Keil, M., Snoeyink, J.: On the time bound for convex decomposition of simple polygons. Int. J. Comput. Geom. Appl. **12**(03), 181–192 (2002)
11. Lingas, A., Pinter, R., Rivest, R., Shamir, A.: Minimum edge length partitioning of rectilinear polygons. In: Proceedings of the Annual Allerton Conference on Communication, Control, and Computing, vol. 10, pp. 53–63 (1982)
12. Lipski, W., Lodi, E., Luccio, F., Mugnai, C., Pagli, L.: On two-dimensional data organization II. Fundam. Informaticae **2**, 245–260 (1979)
13. Lipski, W.: Finding a Manhattan path and related problems. Networks **13**(3), 399–409 (1983)
14. Ohtsuki, T.: Minimum dissection of rectilinear regions. In: Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 1210–1213 (1982)

15. O'Rourke, J., Supowit, K.J.: Some NP-hard polygon decomposition problems. IEEE Trans. Inf. Theory **29**(2), 181–189 (1983)
16. O'Rourke, J., Tewari, G.: The structure of optimal partitions of orthogonal polygons into fat rectangles. Comput. Geom. **28**(1), 49–71 (2004)
17. Poljak, S.: A note on stable sets and colorings of graphs. Comment. Math. Univ. Carol. **15**(2), 307–309 (1974)
18. Uehara, R.: NP-complete problems on a 3-connected cubic planar graph and their applications. Technical report TWCU-M-0004, Tokyo Woman's Christian University (1996)

Compact Self-Stabilizing Leader Election for General Networks

Lélia Blin[1](✉) and Sébastien Tixeuil[2]

[1] Sorbonne Universités, CNRS, Université d'Evry-Val-d'Essonne, LIP6 UMR 7606, 4 place Jussieu, 75005 Paris, France
lelia.blin@lip6.fr

[2] Sorbonne Universités, CNRS, LIP6 UMR 7606, 4 place Jussieu, 75005 Paris, France

Abstract. We present a self-stabilizing leader election algorithm for general networks, with space-complexity $O(\log \Delta + \log\log n)$ bits per node in n-node networks with maximum degree Δ. This space complexity is sub-logarithmic in n as long as $\Delta = n^{o(1)}$. The best space-complexity known so far for general networks was $O(\log n)$ bits per node, and algorithms with sub-logarithmic space-complexities were known for the ring only. To our knowledge, our algorithm is the first algorithm for self-stabilizing leader election to break the $\Omega(\log n)$ bound for silent algorithms in general networks. Breaking this bound was obtained via the design of a (non-silent) self-stabilizing algorithm using sophisticated tools such as solving the distance-2 coloring problem in a silent self-stabilizing manner, with space-complexity $O(\log \Delta + \log\log n)$ bits per node. Solving this latter coloring problem allows us to implement a sub-logarithmic encoding of spanning trees — storing the IDs of the neighbors requires $\Omega(\log n)$ bits per node, while we encode spanning trees using $O(\log \Delta + \log\log n)$ bits per node. Moreover, we show how to construct such compactly encoded spanning trees without relying on variables encoding distances or number of nodes, as these two types of variables would also require $\Omega(\log n)$ bits per node.

1 Introduction

This paper tackles the problem of designing memory efficient self-stabilizing algorithms for the leader election problem. *Self-stabilization* [15] is a general paradigm to provide recovery capabilities to networks. Intuitively, a protocol is self-stabilizing if it can recover from any transient failure, without external intervention. *Leader election* denoted by *L.E.* is one of the fundamental building blocks of distributed computing, as it enables a single node in the network to be distinguished, and thus to perform specific actions. *L.E.* is especially important in the context of self-stabilization as many protocols for various problems assume

This work was performed within Project ESTATE (Ref. ANR-16-CE25-0009-03), supported by French state funds managed by the ANR (Agence Nationale de la Recherche).

that a single leader exists in the network, even after faults occur. Hence, a self-stabilizing *L.E.* mechanism enables such protocols to be run in networks where no leader is given a priori, by using simple stabilization-preserving composition techniques [15]. *Memory efficiency* relates to the amount of information to be sent to neighboring nodes for enabling stabilization. As a result, only mutable memory (used to store variables) is considered for computing memory complexity of a self-stabilizing protocols, while immutable memory (used to store the code of the protocol) is not considered. A small space-complexity induces a smaller amount of information transmission, which (1) reduces the overhead of self-stabilization when there are no faults, or after stabilization [1], and (2) facilitates mixing self-stabilization and replication [21].

An algorithm is *silent* if each of its executions reaches a point in time after which the states of nodes do not change. A non-silent algorithm is said to be *talkative* (see [10]). A foundational result regarding space-complexity in the context of self-stabilizing silent algorithmsis due to Dolev et al. [16], stating that in n-node networks, $\Omega(\log n)$ bits of memory per node are required for solving tasks such as leader election. So, only *talkative* algorithms may have $o(\log n)$-bit space-complexity for self-stabilizing *L.E.* Several attempts to design compact self-stabilizing *L.E.* algorithms (i.e., algorithms with space-complexity $o(\log n)$ bits) were performed but restricted to rings (See e.g. the paper by Blin et al. [10] and references therein). The best result known so far in this context [10] is a deterministic self-stabilizing *L.E.* algorithm for rings of arbitrary size using identifiers of arbitrary polynomially bounded values, with space complexity $O(\log\log n)$ bits per node.

In general networks, self-stabilizing *L.E.* is tightly connected to self-stabilizing tree-construction. On the one hand, the existence of a leader permits time- and memory-efficient self-stabilizing tree-construction [9,12,13,17,25]. On the other hand, growing and merging trees is the main technique for designing self-stabilizing *L.E.* algorithms in networks, as the leader is often the root of an inward tree [2–4]. To the best of our knowledge, all algorithms that do not assume a pre-existing leader [2–4,7] for tree-construction use $\Omega(\log n)$ bits per node. This high space-complexity is due to the implementation of two main techniques, used by all algorithms, and recalled below.

The first technique is the use of a *pointer-to-neighbor* variable, that is meant to designate unambiguously one particular neighbor of every node. For the purpose of tree-construction, pointer-to-neighbor variables are typically used to store the parent node in the constructed tree. Specifically, the parent of every node is designated unambiguously by its identifier, requiring $\Omega(\log n)$ bits for each pointer variable. In principle, it would be possible to reduce the memory to $O(\log \Delta)$ bits per pointer variable in networks with maximum degree Δ, by using node-coloring at distance 2 instead of identifiers to identify neighbors. However, this, in turn, would require the availability of a self-stabilizing distance-2 node-coloring algorithm that uses $o(\log n)$ bits per node. Previous self-stabilizing distance-2 coloring algorithms use variables of large size. For instance, in the algorithm by Herman et al. [22], every node communicates its distance-3 neighborhood to all its neighbors, which yields a space-complexity of $O(\varDelta^3 \log n)$ bits. Johnen et al. [20] draw random colors in the range $[0, n^2]$, which yields a space-complexity of $O(\log n)$ bits. Finally, while the deterministic algorithm of Blair et al. [6] reduces the space-complexity to $O(\log \varDelta)$ bits per node, this is achieved by ignoring the cost of storing another pointer-to-neighbor variable at each node. In absence of a distance-2 coloring (which their algorithm [6] is precisely supposed to produce), their implementation still requires $\varOmega(\log n)$ bits per node. To date, no self-stabilizing algorithm implement pointer-to-neighbor variables with space-complexity $o(\log n)$ bits in arbitrary networks.

The second technique for tree-construction or *L.E.* is the use of a *distance* variable that is meant to store the distance of every node to the elected node in the network. Such distance variable is used in self-stabilizing spanning tree-construction for breaking cycles resulting from arbitrary initial state (see [2–4]). Clearly, storing distances in n-node networks may require $\varOmega(\log n)$ bits per node. There are a few self-stabilizing tree-construction algorithms that are not using explicit distance variables (see, e.g., [14]), but their space-complexity is huge (e.g. $O(n \log n)$ bits [14]). Using the general principle of distance variables with space-complexity below $\varTheta(\log n)$ bits was attempted by Awerbuch et al. [5], and Blin et al. [10]. These papers distribute pieces of information about the distances to the leader among the nodes according to different mechanisms, enabling to store $o(\log n)$ bits per node. However, these sophisticated mechanisms have only been demonstrated in rings. To date, no self-stabilizing algorithms implement distance variables with space-complexity $o(\log n)$ bits in arbitrary networks.

Our results

In this paper, we design and analyze a self-stabilizing *L.E.* algorithm with space-complexity $O(\log \varDelta + \log\log n)$ bits in n-node networks with degree $\varDelta$. This algorithm is the first self-stabilizing *L.E.* algorithm for arbitrary networks with space-complexity $o(\log n)$ (whenever $\varDelta = n^{o(1)}$). It is designed for the standard state model (a.k.a. shared memory model) for self-stabilizing algorithms in networks, and it performs against the unfair distributed scheduler.

The design of our algorithm requires overcoming several bottlenecks, including the difficulties of manipulating pointer-to-neighbor and distance variables using $o(\log n)$ bits in arbitrary networks. Overcoming these bottlenecks was achieved thanks to the development of sub-routine algorithms, each deserving independent special interest described hereafter.

First, we generalize to arbitrary networks the results proposed [10] for rings, and aiming at publishing the identifiers in a bit-wise manner. This generalization allows us to manipulate the identifiers with just $O(\log\log n)$ bits of memory per node denoted in the following by *bits/nd*.

Second, we propose the first *silent* self-stabilizing algorithm for distance-2 coloring that breaks the space-complexity of $\varOmega(\log n)$ *bits/nd*. More precisely this new algorithm achieves a space-complexity of $O(\log \varDelta + \log\log n)$ *bits/nd*. As opposed to previous distance-2 coloring algorithms, we do not use identifiers for encoding pointer-to-neighbor variables, but we use a compact representation of the identifiers to break symmetries. This algorithm allows us to design a compact encoding of spanning trees.

Third, we design a new technique to detect the presence of cycles in the initial configuration resulting from a transient failure. This approach does not use distances, but it is based on the uniqueness of each identifier in the network. Notably, this technique can be implemented by a *silent* self-stabilizing algorithm, with space-complexity $O(\log \Delta + \log\log n)$ *bits/nd.*

Last but not least, we design a new technique to avoid the creation of cycles during the execution of the *L.E.* algorithm. Again, this technique does not uses distances but maintains a spanning forest, which eventually reduces to a single spanning tree rooted at the leader at the completion of the *L.E.* algorithm. Implementing this technique results in a self-stabilizing algorithm with space complexity $O(\log \Delta + \log\log n)$ *bits/nd.*

2 Model and Definitions

2.1 Protocol Syntax and Semantics

We consider a distributed system consisting of n processes that form an arbitrary communication graph. The processes are represented by the nodes of this graph, and the edges represent pairs of processes that can communicate directly with each other. Such processes are said to be *neighbors.*

Let $G = (V, E)$ be an n-node graph, where V is the set of nodes, and E the set of edges and Δ the degree of the graph. A node v has access to a constant unique identifier ID_v, but can only access its identifier one bit at a time, using the $\text{Bit}_v(i)$ function, which returns the position of the i^{th} most significant bit equal to 1 in ID_v. Even though identifiers require $\Omega(\log n)$ *bits/nd* in the worst case, the Bit function can be stored in the immutable code portion of the node. We present here the pseudocode for the Bit_v function at a particular node v. Note that since nodes have unique identifiers, they are allowed to execute unique code. For example, suppose node v has identifier 10 (in decimal notation), or 1010 (in binary notation). Then, one can implement $\text{Bit}_v(i)$ as follows for $v = 1010$:

$$\text{Bit}_v(i) := \begin{cases} 4 & \text{if i} = 1 \\ 2 & \text{if i} = 2 \\ -1 & \text{if } i > 2 \end{cases}$$

Since we assume that all identifiers are $O(\log n)$ bits long, the Bit_v function only returns values with $O(\log\log n)$ bits. Also, when executing Function Bit_v, the program counter only requires $O(\log\log n)$ values. In turn, this position can be encoded with $O(\log\log n)$ bits when identifiers are encoded using $O(\log n)$ bits, as we assume they are. A node v has access to locally unique port numbers associated with its adjacent edges. We do not assume any consistency between port numbers of a given edge. In short, port numbers are constant throughout the execution but initialized by an adversary. Each process contains variables and rules. Variable ranges over a domain of values. The variable var_v denote the variable *var* located at node v. A rule is of the form $\langle label \rangle : \langle guard \rangle \longrightarrow \langle command \rangle$. A *guard* is a boolean predicate over process variables. A *command* is a set of variable-assignments. A command of process p can only update its own variables. On the other hand, p can read the variables of its neighbors. This classical communication model is called the *state model* or the *state-sharing communication model*.

An assignment of values to all variables in the system is called a *configuration*. A rule whose guard is **true** in some system configuration is said to be *enabled* in this configuration. The rule is *disabled* otherwise. The atomic execution of a subset of enabled rules (at most one rule per process) results in a transition of the system from one configuration to another. This transition is called a *step*. A *run* of a distributed system is a maximal alternating sequence of configurations and steps. Maximality means that the execution is either infinite or its final configuration has no rule enabled.

2.2 Schedulers

The asynchronism of the system is modeled by an adversary (a.k.a. *scheduler*) that chooses, at each step, the subset of enabled processes that are allowed to execute one of their rules during this step. Those schedulers can be classified according to their characteristics (like fairness, distribution, ...), and a taxonomy was presented By Dubois *et al.* [18]. Note that we assume here an unfair distributed scheduler. This scheduler is the most challenging since no assumption is made of the subset of enabled processes chosen by the scheduler at each step. We only require this set to be nonempty if the set of enabled processes is not empty in order to guarantee progress of the algorithm.

A *round* is the smallest portion of execution where every process has the opportunity to execute at least one action. In more detail, each process having at least one enabled rule at the beginning of a round r is either scheduled for execution during r, or all its rules become disabled due to neighbors' rules being executed during r.

3 Compact Self-stabilizing Leader Election for Networks

Our new self-stabilizing *L.E.* algorithm is based on a spanning tree-construction rooted at a maximum degree node, without using distances. If multiple maximum degree nodes are present in the network, we break ties with colors and if necessary with identifiers.

Theorem 1. *Algorithm* **C-LE** *solves the* L.E. *problem in a talkative self-stabilizing manner in any n-node graph, assuming the state model and a distributed unfair scheduler, with $O(\log \Delta + \log\log n)$ bits/nd.*

Our talkative self-stabilizing algorithm reuses and extends a technique for obtaining compact identifiers of size $O(\log\log n)$ *bits/nd* presented in Sect. 3.1. Then, the *L.E.* process consists in running several algorithms layers using decreasing priorities:

1. An original silent self-stabilizing distance-2 coloring presented in Subsect. 3.2 that permits to implement pointer-to-neighbors with $o(\log n)$ *bits/nd.*
2. A silent self-stabilizing cycle and illegitimate sub spanning tree-destruction reused from previous work [8,10] presented in Subsect. 3.3.
3. A new silent self-stabilizing cycle detection that does not use distance to the root variables presented in Subsect. 3.4.
4. An original talkative self-stabilizing spanning tree-construction, that still does not use distance to the root variables, presented in Subsect. 3.5. This algorithm is trivially modified to obtain a *L.E.* algorithm.

Due to the lack of space most of the proofs and predicates are delegated in [11].

3.1 Compact Memory Using Identifiers

As many deterministic self-stabilizing *L.E.* algorithms, our approach ends up comparing node unique identifiers. However, to avoid communicating the full $\Omega(\log n)$ bits to each neighbor at any given time, we reuse the scheme devised in previous work [10] to progressively publish node identifiers. Let ID_v be the identifier of node v. We assume that $\text{ID}_v = \sum_{i=0}^{k} b_i 2^i$. Let $I_v = \{i \in \{0, ..., k\}, b_i \neq 0\}$ be the set of all non-zero bit-positions in the binary representation of ID_v. Then, I_v can be written as $\{pos_1, ..., pos_j\}$, where $pos_k > pos_{k+1}$. In the process of comparing node unique identifiers during the *L.E.* algorithm execution, the nodes must first agree on the same bit-position pos_{j-i+1} (for $i = 1, \ldots, j$); this step of the algorithm defines *phase* i. Put differently, the bit-positions are communicated in decreasing order of significance in the encoding of the identifier. In turn, this may propagate it to their neighbors, and possibly to the whole network in subsequent phases. This propagation is used in the following to break symmetries in the coloring problem or to detect a cycle in spanning tree construction.

If all identifiers are in $[1, n^c]$, for some constant $c \geq 1$, then the communicated bit-positions are less than or equal to $c\lceil \log n \rceil$, and thus can be represented with $O(\log \log n)$ bits. However, the number of bits used to encode identifiers may be different for two given nodes, so there is no common upper bound for the size of identifiers. We circumvent this problem using a ranking on bit-positions that is agnostic on the size of the identifiers. We extract of our previous works the part dedicated to the propagation of the identifier bit by bit in phases, remark that we slightly modify our previous work. Since we do not assume that the identifiers of every node are encoded using the same number of bits, simply comparing the i-th most significant bit of two nodes is irrelevant. Instead, we use variable $\widehat{\mathsf{B}}_v$, which represents the most significant bit-position of node v. In other words, $\widehat{\mathsf{B}}_v$ represents the size of the binary representation of ID_v. The variables ph, Bp are the core of the identifier comparison process. Variable ph_v stores the current phase number i, while variable Bp_v stores the bit-position of ID_v at phase i. Remark that the number of non-zero bits can be smaller than the size of the binary representation of the identifier of the node, so if there are no more non-zero bit at phase $i \leq \widehat{\mathsf{B}}_v$, we use $\mathsf{Bp}_v = -1$. To make the algorithm more readable, we introduce variable $\mathcal{C}id_v = (\widehat{\mathsf{B}}_v, \mathsf{ph}_v, \mathsf{Bp}_v)$, called a *compact identifier* in the sequel. When meaningful, we use $\mathcal{C}id_v^i = (\widehat{\mathsf{B}}_v, \mathsf{Bp}_v)$, where $i = \mathsf{ph}_v$.

3.2 Silent Self-stabilizing Distance-2 Coloring

In this section, we provide a new solution to assign colors that are unique up to distance two (and bounded by a polynom of the graph degree) in any graph. Those colors are meant to efficiently implement the pointer-to-neighbor mechanism that otherwise requires $\Omega(\log n)$ *bits/nd.*

Our solution uses compact identifiers to reduce memory usage. When a node v has the same color as (at least one of) its neighbors, then if the node v has the smallest conflicting color in its neighborhood and is not the biggest identifier among conflicting nodes, then v changes its color. To make sure a fresh color is chosen by v, all nodes publish the maximum color used by their neighborhood (including themself). So, when v changes its color, it takes the maximum advertised color plus one. Conflicts at distance two are resolved as follow: let us consider two nodes u and v in conflict at distance two, and let w be (one of) their common neighbor; as w publishes the color of u and v, it also plays the role of a relay, that is, w computes and advertises the maximum identifiers between u and v, using the compact identifiers mechanisms that were presented above; a bit by bit, then, if v has the smallest identifier, it changes its color to a fresh one. To avoid using too many colors when selecting a fresh one, all changes of colors are made modulo an upper bound on the number of neighbors at distance 2, which is computed locally by each node.

Self-stabilizing Algorithm Description. Each node v maintains a color variable denoted by c_v and a degree variable denoted by δ_v. A variable $\check{c}_v$ stores the minimum color in conflict in its neighborhood (including itself). The variable $\widehat{c}_v$ stores the maximum color observed in its neighborhood. We call v a *player* node when v has the minimum color in conflict. Also, we call u a *relay* node when u does not have the minimum color in conflict, yet at least two of its neighbors have the minimum color in conflict.

The rule $\mathbb{R}_\Delta$ assures that the degree variable is equal to the degree of the node. Each node v must maintain its color in range $[1, \Delta(v)^2 + 1]$ to satisfy the memory requirements of our protocol, where $\Delta(v)$ is a function that returns the maximum degree of its neighborhood (including itself). Whenever v's color exceeds its expected range, rule $\mathbb{R}_\Delta^+$ resets the color to one.

Rule $\mathbb{R}_{\mathsf{Up}}$ is dedicated to updating the variables of v whenever they do not match the observed neighborhood of v (see $\mathtt{Bad}(v)$), or when a player node has an erroneous phase variable when comparing its identifier with another player node (see function $\mathtt{Oth}(v)$). In both cases, the v computes the minimum and maximum color and resets its compact identifier variable.

The rule $\mathbb{R}_{\mathtt{Color}}$ increases the color of the node v but maintains the color in some range, when v has the minimum color in conflict and the minimum identifier. The rule $\mathbb{R}_{\mathtt{Bit}}$ increases the phase of v, when v is a player and does not have the minimum identifier at the selected phase. The rule $\mathbb{R}_{\texttt{Relay}}$ updates the identifier variable when v is a relay node.

Algorithm 1. C-Color

```
        R_Δ   : $(\delta_v \neq \deg(v))$ $\longrightarrow \delta_v := \deg(v);$
        R_Δ^+ : $(\delta_v = \deg(v)) \wedge (\mathsf{c}_v > \Delta(v)^2)$ $\longrightarrow \mathsf{c}_v := 1;$
        R_Up  : $(\delta_v = \deg(v)) \wedge (\mathsf{c}_v \leq \Delta(v)^2) \wedge \texttt{Bad}(v)$ $\longrightarrow \mathcal{U}pdate(v);$

R_Color : $(\delta_v = \deg(v)) \wedge (\mathsf{c}_v \leq \Delta(v)^2) \wedge \neg\texttt{Bad}(v) \wedge \texttt{Player}(v) \wedge \texttt{Loser}(v)$
          $\longrightarrow \mathcal{N}ewcolor(v);$

R_Bit   : $(\delta_v = \deg(v)) \wedge (\mathsf{c}_v \leq \Delta(v)^2) \wedge \neg\texttt{Bad}(v) \wedge \texttt{Player}(v) \wedge \neg\texttt{Loser}(v) \wedge \texttt{SPh}^+(v, \texttt{Oth}(v))$
          $\longrightarrow \mathcal{I}ncPh(v);$

R_Relay : $(\delta_v = \deg(v)) \wedge (\mathsf{c}_v \leq \Delta(v)^2) \wedge \neg\texttt{Bad}(v) \wedge \texttt{Relay}(v) \wedge \texttt{RUp}(v, \texttt{PlayR}(v))$
          $\longrightarrow \mathcal{O}pt(v, \texttt{PlayR}(v), \max);$
```

Theorem 2. *Algorithm* **C-Color** *solves the vertex coloration problem at distance two in a silent self-stabilizing manner in graph, assuming the state model, and a distributed unfair scheduler. Moreover, if the n node identifiers are in $[1, n^c]$, for some $c \geq 1$, then* **C-Color** *uses $O(\log \Delta + \log\log n)$ bits/nd. Moreover, algorithm* **C-Color** *stabilizes in $O(n \log^2 n)$ rounds.*

3.3 Cleaning a Cycle or an Impostor-Rooted Spanning Tree

The graph G is supposed to be colored up to distance 2, thanks to our previous algorithm. To construct a spanning tree of G, each node v maintains a variable p_v storing the color of v's parent ($\varnothing$ otherwise). The function $\texttt{Ch}(v)$ to return the subset of v's neighbors considered as its children (that is, each such node u has its p_u variable equal v's color). Note that the variable parent is managed by the algorithm of spanning tree-construction.

An error is characterized by the presence of inconsistencies between the values of the variables of a node v and those of its neighbors. In the process of a tree-construction, an error occurring at node v may have an impact on its descendants. For this reasons, after a node v detects an error, our algorithm cleans v and all of its descendants. The cleaning process is achieved by Algorithm **Freeze**, already presented in previous works [8,10]. Algorithm **Freeze** is run in two cases: cycle detection (thanks to predicate ErCycle(v), presented in Subsect. 3.4), and impostor leader detection (thanks to predicate ErST(v), presented in Subsect. 3.5). An impostor leader is a node that (erroneously) believes that it is a root.

When a node v detects a cycle or an impostor root, v deletes its parent. Simultaneously, v becomes a *frozen* node. Then, every descendant of v becomes frozen. Finally, from the leaves of the spanning tree rooted at v, nodes delete their parent and reset all variables that are related to cycle detection or tree-construction.

So, this cleaning process cannot create a livelock. Algorithm **Freeze** is a silent self-stabilizing algorithm using $O(1)$ bits of memory per node finally from the leaves of the spanning tree rooted in v the nodes delete their parent and reset all the variables relative to the construction of the spanning tree and the detection of the cycle.

It is important to note that a frozen node, or the child of a frozen node, does not participate to cycle detection or spanning tree-construction. It is important to note that a frozen node or a child of a frozen node does not participate to the detection of a cycle or to the construction of the spanning tree.

3.4 Silent Self-stabilizing Algorithm for Cycle Detection

We present in this subsection a self-stabilizing algorithm to detect cycles (possibly due to initial incorrect configuration) without using the classical method of computing the distance to the root. We first present our solution with the assumption of global identifiers (hence using $O(\log n)$ bits for an n-node network) and then using our compact identifier scheme.

Silent Self-stabilizing Algorithm with Identifiers. The main idea to detect cycles is to use the uniqueness of the identifiers. We flow the minimum identifier up to the tree to the root, then if a node whose identifier is minimum receives its identifier, it can detect a cycle. Similarly, if a node v has two children flowing the same minimum identifier, v can detect a cycle. The main issue to resolve is when the minimum identifier that is propagated to the root does not exist in the network (that is, it results from an erroneous initial state).

The variable m_v stores the minimum identifier collected from the leaves to the root up to node v. We denote by E_v the minimum identifier obtained by v during the previous iteration of the protocol (this can be $\varnothing$). A node v may selects among its children the node u with the smallest propagated identifier stored in m_u, we call this child *kid* returned by the function $\mathsf{k}(v)$. Predicate ErCycle(v) is the core of our algorithm. Indeed, a node v can detect the presence of a cycle if it has a parent and if *(i)* one of its child publishes its own identifier, or *(ii)* two of its children publish the same identifier. Let us explain those conditions in more detail. We consider a spanning structure S, a node $v \in S$ and let u and w be two of its children. Suppose that v and u belong to a cycle $\mathtt{C}$, note that, since a node has a single parent, w cannot belong to any cycle (see Fig. 1). Let $\check{\mathsf{m}}$ be the minimum identifier stored by any variable m_z such that z belongs to S. So, z is either in $\mathtt{C}$, or in the subtree rooted to w, denoted by $\mathtt{T}_w$.

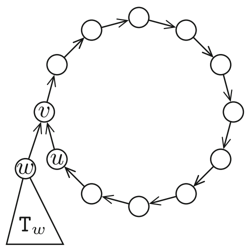


Fig. 1. Spanning structure

First, let us consider the case where $\check{\mathsf{m}}$ is stored in $\mathtt{T}_w$. As any node selects the minimum for flowing the m upstream, there exists a configuration γ where $\mathsf{m}_w = \check{\mathsf{m}}$, and a configuration $\gamma' > \gamma$ where $\mathsf{m}_u = \check{\mathsf{m}}$. In γ', v can detect an error, due to the uniqueness of identifier, it is not possible for two children of v to share the same value when there is no cycle.

Now, let us suppose that $\check{\mathsf{m}}$ is in $\mathtt{C}$, and let v' be the node with the smallest identifier in $\mathtt{C}$, so $\mathsf{m}_{v'} = \check{\mathsf{m}}$ or $\mathsf{m}_{v'} \neq \check{\mathsf{m}}$ ($\mathsf{m}_{v'} \neq \check{\mathsf{m}}$ means that the identifier $\check{\mathsf{m}}$ does not exist in $\mathtt{C}$.) If $\mathsf{m}_{v'} = \check{\mathsf{m}}$, as any node selects the minimum for flowing the m upstream, there exists a configuration γ where $\mathsf{m}_{u'} = \check{\mathsf{m}}$ and u' is the child of v' involved in $\mathtt{C}$, then v' can detect an error. Indeed, due to the uniqueness of identifier, it is not possible that one of its children store its identifier when there is no cycle. The remaining case is when $\mathsf{m}_{v'} \neq \check{\mathsf{m}}$. In this case, as any node selects the minimum for flowing the m upstream, there exists a configuration γ where $\mathsf{m}_z = \check{\mathsf{m}}$, with z belonging to $\mathtt{C}$. When a node v, its parent and one of its children share the same minimum, they restart the computation of the minimum identifier. For this purpose, they put their own identifier in the m variable. To avoid livelock, they also keep track of the previous $\check{\mathsf{m}}$ in variable E_v. Now $\check{\mathsf{m}} = \mathsf{m}_{v'}$, so the system reaches the first case. Note that the variable E_v blocks the live-lock but also the perpetual restart of the nodes, as a result of this, a silent algorithm. Moreover, a node v collects the minimum identifier from the leaves to the root, if m_v contains an identifier bigger than the identifier of the node v, then v detects an error. The same holds, when v has a m_v smaller than m_u with u children of v, since the minimum is computed between $\mathsf{m}_{\mathsf{k}(v)}$ and its own identifier.

$$\begin{aligned}\text{ErCycle}(v) \equiv (\mathsf{p}_v \neq \varnothing) \wedge \Big(& (\mathsf{m}_{\mathsf{k}(v)} = \text{ID}_v) \vee (\exists (u,w) \in \mathtt{Ch}(v) : \mathsf{m}_u = \mathsf{m}_w) \\ & \vee (\mathsf{m}_v > \text{ID}_v) \vee \big((\mathsf{m}_v \neq \text{ID}_v) \wedge (\mathsf{m}_v < \mathsf{m}_{\mathsf{k}(v)})\big)\Big)\end{aligned} \quad (1)$$

Our algorithm only contains three rules. The rule $\mathbb{R}_{\mathtt{Min}}(v)$ updates the variable m_v if the variable m_u of a child u is smaller, nevertheless this rule is enabled if and only if the variable E_v does not contain the minimum m_u published by the child. When v and its relatives have the same minimum, v declares its intent to restart a minimum identifier computation by erasing its current (and storing it in E_v). The rule $\mathbb{R}_{\mathtt{Start}}(v)$ is dedicated to declaring its intent to restart. When all its neighbors have the same intent, v can restart (see rule $\mathbb{R}_{\mathtt{ID}}(v)$).

Algorithm 2. Algorithm **Break** For node v with $\neg$ErCycle(v)

```
$\mathbb{R}_{\mathtt{Min}}$   : $(\mathsf{m}_v > \mathsf{m}_{\mathsf{k}(v)}) \wedge (\mathsf{E}_v \neq \mathsf{m}_{\mathsf{k}(v)})$                 $\longrightarrow \mathsf{m}_v := \mathsf{m}_{\mathsf{k}(v)}$;
$\mathbb{R}_{\mathtt{Start}}$ : $(\mathsf{m}_{\mathsf{p}_v} = \mathsf{m}_v = \mathsf{m}_{\mathsf{k}(v)}) \wedge (\mathsf{E}_v \neq \mathsf{m}_v)$         $\longrightarrow \mathsf{E}_v := \mathsf{m}_v$;
$\mathbb{R}_{\mathtt{ID}}$    : $(\mathsf{E}_{\mathsf{p}_v} = \mathsf{E}_v = \mathsf{E}_{\mathsf{k}(v)} = \mathsf{m}_v) \wedge (\mathsf{m}_v \neq \text{ID}_v)$ $\longrightarrow \mathsf{m}_v := \text{ID}_v$;
```

Theorem 3. *Algorithm* **Break** *solves the detection of cycle in n-node graph in a silent self-stabilizing manner, assuming the state model, and a distributed unfair scheduler. Moreover, if the n node identifiers are in $[1, n^c]$, for some $c \geq 1$, then algorithm* **Break** *uses $O(\log n)$ bits/nd.*

3.5 Talkative Spanning Tree-Construction Without Distance

Our approach for self-stabilizing *L.E.* is to construct a spanning tree whose root is to be the elected leader. Two main obstacles to self-stabilizing tree-construction are the possibility of an arbitrary initial configuration containing one or more cycles, or the presence of one or more impostor-rooted spanning trees. We already explained how the cycle detection and cleaning process takes place, so we focus in this section on *cycleless* configurations.

The main idea is to mimics the *fragments* approach introduced by Gallager *et al.* [19]. In an ideal situation, at the beginning each node is a fragment, each fragment merges with a neighbor fragment holding a bigger root signature, and at the end remains only one fragment, rooted in the root with the biggest signature (that is, the root with maximum degree, maximum color, and maximum global identifier). To maintain a spanning structure, the neighbors that become relatives (that is, parents or children) remain relatives after that. Note that the relationship may evolve through time (that is, a parent can become a child and vice versa). So our algorithm maintains that as an invariant.

Indeed, when two fragments merge, the one with the root with smaller signature F_1 and the other one with a root with bigger signature F_2, the root of F_1 is re-rooted toward its descendants until reaching the node that identified F_2. This approach permits to construct an acyclic spanning structure, without having to maintain distance information.

The variable R_v stores the signature relative to the root (that is, its degree, its color, and its identifier). Note that, the comparison between two R is done using lexical ordering. The variable new_v stores the color of the neighbor w of v leading to u with $\mathsf{R}_u > \mathsf{R}_v$ if there exists such a node, and $\varnothing$ otherwise. The function $f(v)$ returns the color of the neighbor of v with the maximum root.

Theorem 4. *Algorithm* **ST** *solves the spanning tree-construction problem in a silent self-stabilizing way in any n-node graph, assuming the absence of spanning cycle, the state model, and a distributed unfair scheduler, using $O(\log n)$ bits/nd.*

We adapt **ST** to use compact identifiers and obtain Algorithm **C-ST**. It is simple to compare two compact identifiers when the nodes are neighbors. Along the algorithm execution, some nodes become non-root, and therefore the remaining root of fragments can be far away, separated by non-root nodes. To enable multi-hop comparison, we use a broadcasting and convergecast wave on a spanning structure to assure the propagation of the compact identifier.

Theorem 5. **C-ST** *solves the spanning tree-construction problem in a talkative self-stabilizing way in any n-node graph, assuming the absence of spanning cycle, the state model, and a distributed unfair scheduler, in $O(\log \Delta + \log\log n)$ bits/nd.*

4 Self-stabilizing Leader Election

We now present the final assembly of tools we developed to obtain a self-stabilizing *L.E.* algorithm. We add to Algorithm **C-ST** an extra variable ℓ that is maintained as follows: if a node v has no parent, then $\ell_v = true$, otherwise, $\ell_v = false$. Variable ℓ_v is meant to be the output of the *L.E.* process.

Proof sketch of Theorem 1: Our self-stabilizing *L.E.* algorithm results from combining severals algorithms. As already explained, a higher priority algorithm resets all the variables used by lesser priority algorithms. Moreover, lesser priority algorithm do not modify the variables of the higher priority algorithms. Algorithms are prioritized as follows: **C-Color**, **Freeze**, **C-Break** and **C-ST**. Only algorithm **C-ST** is talkative. We first proove that the number of activations of rules of algorithm **C-ST** are bounded if there exist nodes enabled by **C-Color**, **Freeze** or **C-Break**, ensuring termination of those components. Thanks to Theorem 5, we obtain a spanning tree rooted in the node with the maximum degree, maximum color, and maximum identifier. As a consequence, only the root r has $\ell_r = true$ and every other node $v \in V \setminus \{r\}$ has $\ell_v = false$.

5 Conclusion

We presented the first self-stabilizing leader election for arbitrary graphs of size n that uses $o(\log n)$ bits of memory per node, breaking a long-standing lower bound. Our solution does not require any weakening of the usual self-stabilization model. Besides tree construction and leader election, our research paves the way for new memory efficient self-stabilizing algorithms. For example, some of the solutions for self-stabilizing maximal matching construction use a fixed number of "pointer to neighbor" variables [23,24]. Using our distance two coloring process would permit to go from $O(\log n)$ to $O(max\{\log \Delta, \log\log n\})$ bits per node.

References

1. Adamek, J., Nesterenko, M., Tixeuil, S.: Evaluating practical tolerance properties of stabilizing programs through simulation: the case of propagation of information with feedback. In: Richa, A.W., Scheideler, C. (eds.) SSS 2012. LNCS, vol. 7596, pp. 126–132. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33536-5_13
2. Afek, Y., Bremler-Barr, A.: Self-stabilizing unidirectional network algorithms by power supply. Chicago J. Theor. Comput. Sci. (1998)
3. Afek, Y., Kutten, S., Yung, M.: Memory-efficient self stabilizing protocols for general networks. In: van Leeuwen, J., Santoro, N. (eds.) WDAG 1990. LNCS, vol. 486, pp. 15–28. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-54099-7_2
4. Arora, A., Gouda, M.G.: Distributed reset. IEEE Trans. Comput. **43**(9), 1026–1038 (1994)
5. Awerbuch, B., Ostrovsky, R.: Memory-efficient and self-stabilizing network reset. In: PODC, pp. 254–263. ACM (1994)
6. Blair, J.R.S., Manne, F.: An efficient self-stabilizing distance-2 coloring algorithm. Theor. Comput. Sci. **444**, 28–39 (2012)
7. Blin, L., Boubekeur, F., Dubois, S.: A self-stabilizing memory efficient algorithm for the minimum diameter spanning tree under an omnipotent daemon. In: IPDPS 2015, pp. 1065–1074 (2015)

8. Blin, L., Fraigniaud, P.: Space-optimal time-efficient silent self-stabilizing constructions of constrained spanning trees. In: Proceedings of ICDCS 2015, pp. 589–598 (2015)
9. Blin, L., Potop-Butucaru, M., Rovedakis, S.: A super-stabilizing log(n)log(n)-approximation algorithm for dynamic steiner trees. Theor. Comput. Sci. **500**, 90–112 (2013)
10. Blin, L., Tixeuil, S.: Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. Distrib. Comput. 1–28 (2017). https://doi.org/10.1007/s00446-017-0294-2
11. Blin, L., Tixeuil, S.: Compact self-stabilizing leader election for arbitrary networks. Technical report 1702.07605, ArXiv eprint, Febrary 2017
12. Chen, N.S., Yu, H.P., Huang, S.T.: A self-stabilizing algorithm for constructing spanning trees. Inf. Process. Lett. **39**(3), 147–151 (1991)
13. Collin, Z., Dolev, S.: Self-stabilizing depth-first search. Inf. Process. Lett. **49**(6), 297–301 (1994)
14. Delaët, S., Ducourthial, B., Tixeuil, S.: Self-stabilization with r-operators revisited. J. Aerosp. Comput. Inf. Commun. (JACIC) **3**(10), 498–514 (2006)
15. Dolev, S.: Self-stabilization. MIT Press, Cambridge (2000)
16. Dolev, S., Gouda, M.G., Schneider, M.: Memory requirements for silent stabilization. Acta Inf. **36**(6), 447–462 (1999)
17. Dolev, S., Israeli, A., Moran, S.: Self-stabilization of dynamic systems assuming only read/write atomicity. Distrib. Comput. **7**(1), 3–16 (1993)
18. Dubois, S., Tixeuil, S.: A taxonomy of daemons in self-stabilization. Technical report 1110.0334, ArXiv eprint, October 2011
19. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Program. Lang. Syst. **5**(1), 66–77 (1983)
20. Gradinariu, M., Johnen, C.: Self-stabilizing neighborhood unique naming under unfair scheduler. In: Sakellariou, R., Gurd, J., Freeman, L., Keane, J. (eds.) Euro-Par 2001. LNCS, vol. 2150, pp. 458–465. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44681-8_67
21. Herman, T., Pemmaraju, S.V.: Error-detecting codes and fault-containing self-stabilization. Inf. Process. Lett. **73**(1–2), 41–46 (2000)
22. Herman, T., Tixeuil, S.: A distributed TDMA slot assignment algorithm for wireless sensor networks. In: Nikoletseas, S.E., Rolim, J.D.P. (eds.) ALGOSENSORS 2004. LNCS, vol. 3121, pp. 45–58. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27820-7_6
23. Inoue, M., Ooshita, F., Tixeuil, S.: An efficient silent self-stabilizing 1-maximal matching algorithm under distributed daemon without global identifiers. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 195–212. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49259-9_17
24. Inoue, M., Ooshita, F., Tixeuil, S.: An efficient silent self-stabilizing 1-maximal matching algorithm under distributed daemon for arbitrary networks. In: Spirakis, P., Tsigas, P. (eds.) SSS 2017. LNCS, vol. 10616, pp. 93–108. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69084-1_7
25. Korman, A., Kutten, S., Masuzawa, T.: Fast and compact self stabilizing verification, computation, and fault detection of an MST. In: Proceedings of PODC 2011, pp. 311–320. ACM, New York (2011)

Random Walks with Multiple Step Lengths

Lucas Boczkowski[1], Brieuc Guinard[1(✉)], Amos Korman[1(✉)], Zvi Lotker[2], and Marc Renault[3]

[1] IRIF, CNRS and University Paris Diderot, Paris, France
{guinard,pandit}@irif.fr
[2] Ben Gurion University of the Negev, Beersheba, Israel

[3] Computer Sciences Department, University of Wisconsin - Madison, Madison, USA

Abstract. In nature, search processes that use randomly oriented steps of different lengths have been observed at both the microscopic and the macroscopic scales. Physicists have analyzed in depth two such processes on grid topologies: *Intermittent Search*, which uses two step lengths, and *Lévy Walk*, which uses many. Taking a computational perspective, this paper considers the number of distinct step lengths k as a *complexity measure* of the considered process. Our goal is to understand what is the optimal achievable time needed to cover the whole terrain, for any given value of k. Attention is restricted to dimension one, since on higher dimensions, the simple random walk already displays a quasi linear cover time.

We say X is a k-*intermittent search* on the one dimensional n-node cycle if there exists a probability distribution $\mathbf{p} = (p_i)_{i=1}^{k}$, and integers $L_1, L_2, \ldots, L_k$, such that on each step X makes a jump $\pm L_i$ with probability p_i, where the direction of the jump ($+$ or $-$) is chosen independently with probability 1/2. When performing a jump of length L_i, the process consumes time L_i, and is only considered to visit the last point reached by the jump (and not any other intermediate nodes). This assumption is consistent with biological evidence, in which entities do not search while moving ballistically.

We provide upper and lower bounds for the cover time achievable by k-intermittent searches for any integer k. In particular, we prove that in order to reduce the cover time $\Theta(n^2)$ of a simple random walk to linear in n up to logarithmic factors, roughly $\frac{\log n}{\log \log n}$ step lengths are both necessary and sufficient, and we provide an example where the lengths form an exponential sequence.

In addition, inspired by the notion of intermittent search, we introduce the *Walk or Probe* problem, which can be defined with respect to arbitrary graphs. Here, it is assumed that querying (probing) a node takes significantly more time than moving to a random neighbor. Hence, to efficiently probe all nodes, the goal is to balance the time spent walking randomly and the time spent probing. We provide preliminary results for connected graphs and regular graphs.

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 648032).

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 174–186, 2018.
https://doi.org/10.1007/978-3-319-77404-6_14

1 Introduction

1.1 Background and Motivation

The theory of random walks was first studied in an attempt to abstract the movement of natural entities, such as particles or insects [16,34]. The term "random walk" itself was originally coined by Pearson in 1905 aiming to model the movement of a mosquito in a forest [34]. Random walk theory has since attracted the attention of researchers in many different disciplines, and has developed into one of the most impressive manifestations of a cross-disciplinary theory. In particular, motivated by the simplicity of this memoryless algorithm, mathematicians and computer scientists have studied random walks intensively, focusing mostly on analyzing its time complexities in finite graphs [2,12,17,18,28,29], as well as on identifying various applications of it in different, often seemingly unrelated, computational contexts, see e.g. [3,22,35] among many others.

In the last couple of decades, empirical evidence has suggested that in various natural contexts, movement appears to be similar to random walks, yet with heterogeneous step lengths (jumps). Examples appear both on the microscopic scale, such as in the reaction pathway of DNA binding proteins, immune cells movement, crawling amoeba, optics, and in low dimension Hamiltonian chaos [8,13,21,25], as well as on the macroscopic scale, such as in albatrosses, bumblebees, deer, and even humans [4,5,9,26,36,37,39,40][1]. Most of these examples appear in search contexts, e.g., searching for pathogens or food. It has been further argued in these works that in the corresponding contexts, biological entities alternate between slow diffusing phases in which targets can be detected, and faster phases of ballistic movements (which are typically more rare) during which the search efficiency is weak, effectively allowing targets to be found only between jumps. This compromise between moving and searching has also been studied in deterministic settings [14]. From a search efficiency perspective, it has further been argued that such processes can help to strike a proper balance between global exploration and local exploitation.

Within this family of search strategies, two extreme cases have been extensively studied, namely, *Intermittent search* and *Lévy walks* (see the survey [6] and the references therein). The former process is essentially a random walk with two step lengths: choose an angle uniformly at random (u.a.r), then take a step of unit length with some probability p; otherwise, take a step of some predetermined larger length L. In the latter process, step lengths have a probability distribution that is heavy-tailed: at each step an angle is chosen u.a.r, and the probability to perform a step of length d is proportional to $d^{-\alpha}$, for some fixed parameter $\alpha > 1$. Based on differential equation techniques, these two types of processes have been studied by physicists, aiming to optimize the parameters involved in order to minimize the hitting time under various target distributions in continuous Euclidean spaces [6,10,30,31,33,40]. For example, [31,33] showed

[1] Some of these statistical findings which claim that these animals perform Lévy walks have recently been under debate, due to the difficulty of fitting empirical data to a particular distribution of step lengths [15,38].

that on the one dimensional n-cycle of length n, an intermittent search with the right choice of parameters can reduce the cover time of a simple random walk from roughly n^2 to roughly $n^{4/3}$. Lévy walks can reduce the cover time substantially further. Indeed, with a little extra work (as appears in the full version), it follows from [1] that a Lévy walk process with parameter $\alpha = 2$ can reduce the cover time to almost linear (up to polylog factors).

This paper studies random walk processes with multiple step lengths from a more unified computational perspective. Specifically, driven by the plausible assumption that utilizing more step lengths (while associating to each a tailored probability) may require more computational resources, our main subject of interest concerns quantifying the trade-off between the number of step lengths and the best possible search performances. Specifically, we are interested in the best cover time achievable by a random walk that uses k step lengths for every integer k. From a technical point of view, the challenge lies in understanding what is the best possible balance between actions on different scales, ranging from highly local ones (small jumps) to highly global ones (large jumps) [24,27].

The underlying topology we concentrate on is a discrete cycle of n nodes. It is possible to define the notion of random walks with multiple step lengths in tori or grids of all dimensions, but in this preliminary work, we focus on one dimension since this case enjoys the highest increase in performance as the number of step lengths grows. Indeed, simple random walks are already highly competitive in higher dimensions. We note that one dimension search finds relevance in several biological contexts, including in collective navigation by ants [19] and in the reaction pathway of DNA binding proteins [8,13]. The latter case is a good example of a search with two phases, one which is three-dimensional and fast, the other one-dimensional, slow, which corresponds to sliding along the DNA to find the target site.

Finally, inspired by the notion of intermittent random walks, and aiming to further develop the aforementioned balancing principle, we introduce and investigate a problem, called *Walk or Probe*, which can be defined with respect to arbitrary underlying graphs, and is of independent interest. Here, it is assumed that querying (probing) a node takes significantly more time than moving to a random neighbor. This assumption is consistent with the aforementioned hypothesis that many processes in nature, including e.g., immune cells, cannot engage in moving fast, and, at the same time, execute their search mechanism. Hence, to probe all nodes in a relatively short time, the goal is to balance the time spent walking randomly and the time spent probing. In some sense, a long phase in which the process executes a random walk may be interpreted as "implementing a long jump", in the sense that, with a certain cost, it allows the process to re-start at a different area of the graph.

1.2 Models

k-intermittent search. Let C_n be the n-node cycle and let k be an integer. We say X is a *k-intermittent search* on C_n if there exists a probability distribution $\mathbf{p} = (p_i)_{i=1}^{k}$, where $\sum_i p_i = 1$, and integers $L_1, L_2, \ldots, L_k$ such that, on each step,

X makes a jump $\{0, -L_i, +L_i\}$ with probability respectively $p_i/2, p_i/4, p_i/4$. Overall, with probability $1/2$, the process X stays in place[2]. The numbers (p_i) and (L_i) are called the *parameters* of the search process X.

Our goal is to show upper and lower bounds on the *cover time* of a k-intermittent search; that is, the expected time to visit every node of the ambient graph C_n, where we assume that a jump from some point to b visits only the endpoint b, and not any of the intermediate nodes. Importantly, we are interested in time rather than the number of moves and, hence, need to account for the travel time of jumps. For simplicity, we assume that the speed of the walker is constant (rather than varying between step lengths), which in particular means that it takes one unit of time to make a move to a neighbor and L units of time to make a jump of length L.

More formally, let us denote by $V_1, V_2, \ldots, V_s, \ldots$ independent random variables taking value L_i with probability p_i for every $i \in [k]$. We also use sign variables $\xi_1, \xi_2, \ldots$ which take value 0 or ± 1 with probability $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$. We call a product $\xi_s V_s$ a *jump* and ξ_s is the *sign* of the jump. We can then define the move-process $Z(m)$ on $\mathbb{Z}$ and $X(m)$ on the cycle C_n, after m moves, as

$$Z(m) = \sum_{s=1}^{m} \xi_s \cdot V_s, \qquad\qquad X(m) = Z(m) \mod n. \tag{1}$$

As we consider it takes one unit of time to travel a distance 1, the time it took to accomplish the first m moves, denoted $T(m)$, is defined as

$$T(m) := \sum_{s=1}^{m} |\xi_s| \cdot V_s. \tag{2}$$

On the finite graph C_n, we denote by M the random number of moves needed before X has visited every node of C_n. The quantity whose expectation we want to bound is $T(M)$, the time needed to visit all nodes, which is called the *cover time*.

Walk or Probe. Consider a simple random walker that walks on a connected graph G and aims to *probe* all nodes in G as fast as possible. The walker at a node is unable to detect whether it has previously probed it. At this point it needs to decide whether to continue the walk or probe it and then continue the walk. Crucially, probing a node is time consuming, and can potentially be very slow with respect to the time required to move between neighbors. Specifically, let us assume that each edge traversal costs 1 unit of time, while probing a node costs $C \geq 0$ time units, where C can be a function of several parameters of G (e.g., the number of nodes, edges, or maximal degree).

The Walk or Probe problem aims to find a strategy that balances the time spent in walking vs. probing so as to minimize the *probing cover time*, that is, the expected time until all nodes are probed.

[2] This laziness assumption is used for technical reasons, as is common in many other contexts of random walks. However, note with Eq. (2) that this assumption does not affect the time of the process.

1.3 Our Results

k-intermittent search on the cycle. We report our results in terms of cover time, but the same bounds, divided by a $\log n$ term, apply for hitting times (the time to find any given node), as is clear from our proofs.

Definition 1. *Let B be an integer. Define the k-*intermittent search *with base B, by the parameters*[3] $L_i = B^i$ *for every* $1 \leq i \leq k-1, p_i = \frac{1}{L_i}$ *and* $L_0 = 1$ *and* $p_0 = 1 - \sum_{i=1}^{k-1} p_i$.

Theorem 1. *Let k, B, n be integers such that $2 \leq B < n$ and $B^{k-1} \leq n \leq B^k$. The cover time of the k-intermittent search with base B on the n-cycle is at most $poly(k) \cdot poly(B) \cdot n \log n$.*

Hence, from Theorem 1, taking $B = \lceil n^{1/k} \rceil$, we derive the following corollary.

Corollary 2. *For any $k \leq \frac{\log n}{\log \log n}$, there exists a k-intermittent search with cover time $n^{1+O(\frac{1}{k})} \log n$. In particular, if $k = \frac{\log n}{\log \log n}$, then the expected cover time is $n \cdot \log^{O(1)}(n)$.*

Corollary 2 is almost tight, as shown by the following lower bound:

Theorem 3. *For every $\varepsilon > 0$, there exist sufficiently small constants $c, c' > 0$ such that for $k \leq c' \frac{\log n}{\log \log n}$, any k-intermittent search cannot achieve a cover time better than $c \cdot n^{1+\frac{1/2-\varepsilon}{k+1}}$. In particular, for $k = o(\frac{\log n}{\log \log n})$, the cover time is $n \cdot \log^{\omega(1)}(n)$.*

Walk or Probe. Consider the Walk or Probe model on a connected graph $G = (V, E)$ with n nodes, with cost of probing C. Denote by t_{cov} (resp. t_{mix}) the cover time (resp. mixing time) of a random walk on G (see Sect. 4 for the definition of the mixing time). The most naive strategy is to probe after each step, in which case we get a probing cover time of:

$$(C+1) \cdot t_{cov}. \tag{3}$$

We show that there exists another simple strategy whose running time depends on t_{mix}, which can be much more efficient in some cases. Let us call *t-strategy* the tactic that consists in probing only once every t steps. Let $G = (V, E)$ be an n-node connected graph.

[3] Note that $\sum_{i=1}^{k-1} p_i = \sum_{i=1}^{k-1} B^{-i} = \frac{1}{B-1}(1 - \frac{1}{B^{k-1}})$ is a decreasing function of $B \geq 2$, and for $B \geq 3$ is less than $\frac{1}{2}$ so that $p_0 \geq \frac{1}{2}$. For $B = 2$, we would have $p_0 = \Theta(\frac{1}{n})$. In this case, we can change the definition of p_i by dividing by a factor 2, which will allow to have $p_0 \geq \frac{1}{2}$. This does not change anything beyond the constants we use in the proofs and in the results.

Theorem 4. *The probing cover time of the $\sqrt{t_{mix}}$-strategy is*

$$O\left((C+\sqrt{t_{mix}})\cdot|E|\log n\right). \tag{4}$$

And if G is regular, the probing cover time of the $\sqrt{t_{mix}}$-strategy is

$$O\left((C+\sqrt{t_{mix}})\cdot n\log n\right). \tag{5}$$

In particular, we can derive from Theorem 4 (taking $C=0$) the following upper bound.

Corollary 5. *For any n-node connected graph $G=(V,E)$,*

$$t_{cov}=\begin{cases}O(\sqrt{t_{mix}}n\log n) & \text{if } G \text{ is regular}\\ O(\sqrt{t_{mix}}|E|\log n) & \text{otherwise.}\end{cases} \tag{6}$$

For the case of regular graphs, the upper bound on t_{cov} mentioned in Corollary 5 was already known ([23, Theorem 1.4]). To the best of our knowledge, the latter bound, for general graphs, is new.

Since the search should cover every node and probe every node of G, we have the following trivial lower bound for the probing cover time:

$$t_{cov}+C\cdot n. \tag{7}$$

Let us compare this lower bound to the upper bounds on the probing cover time of the 1- and $\sqrt{t_{mix}}$-strategies, given by Eqs. (3) and (5), in the case of regular graphs.

One family of instances consists of regular graphs for which Eq. (6) is tight (up to poly-logarithmic factors). In this case Eq. (5) becomes, up to poly-logarithmic factors, $Cn+t_{cov}$, and so the $\sqrt{t_{mix}}$-strategy is near optimal. This happens, for example, if the mixing time is poly-logarithmic (e.g. cliques and all expander graphs, such as random r-regular graphs [20]) and in such cases, by Eq. (3), the 1-strategy is also near optimal since the cover time is almost linear as Eq. (6) shows. On the other hand, Eq. (6) is also tight for the path and the cycle, on which the mixing time and cover time are $O(n^2)$. Hence, in this case the $\sqrt{t_{mix}}$-strategy significantly outperforms the trivial 1-strategy when C is large.

If Eq. (6) is not tight then the $\sqrt{t_{mix}}$-strategy may not be optimal. An example where this happens is the torus of dimension $d\geq 2$: Indeed, the mixing time of such a torus is $\Theta(n^{2/d})$, while the cover time is almost linear [29, Sects. 5.3.2 and 10.4]. In this case the 1-strategy has probing cover time near optimal by Eq. (3), but the $\sqrt{t_{mix}}$-strategy, which needs at least $\sqrt{t_{mix}}n$ steps (because it must probe at least n times), is not optimal.

Finally, regarding the general bound in Eq. 4 (for not-necessarily regular graphs), let us look at Erdos-Renyi random graphs $\mathcal{G}(n,p)$ with $p=c/n$ for a constant $c>1$. In this case the mixing time is a.a.s. $\Theta(\log^2 n)$ [7, Theorem 1.1] and the cover time is a.a.s. $\Theta(n\log^2 n)$ [11, Theorem 2(a)], and the number of vertices is a.a.s. linear in n. Therefore, by Eq. (4), the $\sqrt{t_{mix}}$-strategy has probing cover time $O(Cn\log n+n\log^2 n)$, while the 1-strategy, by Eq. (3), takes time $(C+1)n\log^2 n$, and hence the former strategy gains a logarithmic factor in the number of probes.

1.4 Preliminaries

Notation. When writing logarithms, unless mentioned otherwise, we assume that the base is 2. We denote the cycle of length n by C_n and label its nodes with the serials $\{0, 1, \dots, n-1\}$. We use the symbols c, c', c'' for constants that we do not compute explicitly. In order to avoid the use of too many symbols, we sometimes employ the same symbol for different constants.

From counting time to counting steps. Recall that $T(m) = \sum_{s=1}^{m} |\xi_s| \cdot V_s$. Let us recall also the well-known.

Lemma 6 (Wald's identity). If $(X_t)_{t\geq 0}$ is an i.i.d. sequence with $|E(X_0)| < \infty$, and T is a stopping time for this sequence (i.e. at each time t, the occurrence of the event $T = t$ depends only on $X_0, X_1, \dots, X_t$), with $|\mathbb{E}(T)| < \infty$, then:

$$\mathbb{E}(\sum_{t=0}^{T} X_t) = \mathbb{E}(T)\mathbb{E}(X_0).$$

As a direct application, we obtain the following.

Lemma 7. $\mathbb{E}(T(M)) = \mathbb{E}(M) \cdot \frac{\mathbb{E}(V_1)}{2}$.

From hitting times to cover time. The following relates the probability to hit a node to the cover time, and can be thought of as a loose but easy Matthew (upper) bound.

Lemma 8. Let $(X_t)_t$ be a finite Markov chain with n states and, for a state $x, M(x)$ be the random number of steps before the chain hits x. If there are m and $p > 0$ such that for every state $x, \Pr(M(x) \leq m) \geq p$, then the cover time of the chain is $O(\frac{m \log(n)}{p})$.

Proof. Split the moves into phases, each composed of m consecutive moves. Using the Markov property of the process, during each phase, x is visited with probability at least p, independently of the trajectory on previous phases. Then the probability to not have visited x after ℓ phases yet, is less than $(1-p)^\ell$. Using a union bound, the probability that there exists a node which has not been visited after ℓ phases is thus less than $n(1-p)^\ell$. For

$$\ell_j = \frac{-\log(2^j n)}{\log(1-p)} = O\left(\frac{j \log n}{p}\right),$$

this probability is less than 2^{-j}. Hence, the expected number of phases before covering all nodes is less than

$$\sum_j \ell_{j+1} \cdot 2^{-j} = O\left(\frac{\log n}{p}\right).$$

Multiplying this number by m gives the expected number of moves before covering all states. □

From the infinite line to the n-nodes cycle

Remark 9. With the notations of Eq. (1), if for $x \in [0, n-1] \subset \mathbb{Z}, M_{\mathbb{Z}}(x)$ (resp. $M_{C_n}(x)$) is the random number of moves for Z (resp $X = Z \mod n$) to hit x, we have $M_{C_n}(x \mod n) \leq M_{\mathbb{Z}}(x)$.

Combining this remark with the two previous lemmas, we obtain:

Claim 10. If we have m and p such that for all $x \in [0, n-1], \Pr(M_{\mathbb{Z}}(x) \leq m) \geq p$, then the cover time of X is $O\left(\frac{\mathbb{E}(V_1) m \log n}{p}\right)$. Note that $\mathbb{E}(V_1) = k$ in the particular base B process with k lengths.

A useful identity. All of our upper bounds rely on the following identity, the usefulness of which was highlighted in [23]. If N is a nonnegative random variable then:

$$\Pr(N \geq 1) = \frac{\mathbb{E}(N)}{\mathbb{E}(N \mid N \geq 1)}. \tag{8}$$

2 Upper Bound on the Cycle

This section is dedicated to proving Theorem 1. We begin with an overview of the proof. Using Claim 10, the bound of Theorem 1 can be established by studying the process on the infinite line.

The core of the computations for Theorem 1 are encapsulated in Lemmas 11 and 12. These bounds on the distribution of the move process Z are used to lower bound the expected number of visits to any point on the infinite line, and upper bound the expected number of returns to the starting point. A lower bound on the probability of any node being visited follows from Eq. (8).

The following two lemmas consider the move process on the infinite line and present the main technical aspects of the proof. The corresponding proofs are all deferred to the full version. Recall, from Theorem 1, that we assume that k is such that $k \geq 2$ and $B^{k-1} \leq n \leq B^k$.

Lemma 11. There exist sufficiently large constants c and c' (independent of all other parameters), such that for any $j \in [0, k-1]$ and $m \geq cB^{j+1} \log kB$, we have $\Pr(Z(m) = 0) \leq c' B^{-j}$.

Lemma 12. Let c (resp. c') be a big (resp. small) enough constant. Let $m_0 := cB^{k+1} \log kB$. For any $x \in [0, n-1] \subseteq [0, B^k - 1]$, and $m \in [m_0, m_0 + n]$, we have $\Pr(Z(m) = x) \geq \frac{c'}{B^k \sqrt{\log kB}}$.

With Claim 10, Lemmas 11, and 12, we are now ready to prove Theorem 1.

Proof (Proof of Theorem 1). Fix a point $x \in \mathbb{Z}$. Define $N_x(m)$ as the number of visits to x after the first m moves have been made. We want to apply Eq. (8) to

$N_x(m)$. This will imply a lower bound on the probability that the node x has been visited before m moves (i.e., the event $N_x(m) \geq 1$). Set

$$m_0 = cB^{k+1} \log kB, \text{ and } m_1 = m_0 + n,$$

for some constant $c > 0$ big enough so that both the conclusion of Lemmas 11 and 12 hold. Note that since $B^{k-1} \leq n, m_1 = O(nB^2 \log kB)$.

Our first goal is to show that $\mathbb{E}(N_x(m_1) \mid N_x(m_1) \geq 1) = O(kB^2 \log kB)$. Since the number of returns to x before time m_1 is maximized when we begin at x, we have:

$$\mathbb{E}(N_x(m_1) \mid N_x(m_1) \geq 1) \leq \mathbb{E}(N_0(m_1)) \leq 1 + \sum_{m=1}^{m_1} \Pr(Z(m) = 0). \tag{9}$$

Next, using Lemma 11, for any $j \in [1, k-1]$ and any m, such that

$$cB^{j+1} \log kB \leq m < cB^{j+2} \log kB,$$

we upper bound $\Pr(Z(m) = 0)$ by $c'B^{-j}$ with c' some other constant. When $m \leq cB^2 \log kB$, we use $\Pr(Z(m) = 0) \leq 1$. Using Inequality (9), we obtain:

$$\begin{aligned}\mathbb{E}(N_x(m_1) \mid N_x(m_1) \geq 1) &= O\left(B^2 \log kB + \sum_{j=1}^{k-2} \frac{B^{j+2} \log kB}{B^j}\right) \\ &= O(kB^2 \log kB)\ , \end{aligned} \tag{10}$$

as desired. We next lower bound $\mathbb{E}\left(N_x(m_1)\right)$ for any $x \in [0, n-1]$, by summing the Inequality of Lemma 12 between m_0 and m_1:

$$\mathbb{E}(N_x(m_1)) \geq \frac{c' \cdot n}{B^k \sqrt{\log kB}} = \Omega\left(\frac{1}{B\sqrt{\log kB}}\right). \tag{11}$$

Dividing (11) by (10), it follows from Eq. (8) that for any $x \in [0, n-1]$, we have $\Pr(N_x(m_1) > 1) = \Omega\left(\frac{1}{kB^3 \log^{3/2} kB}\right)$. Using Claim 10 with $m = m_1 = O(nB^2 \log kB)$ and $p = \Omega\left(\frac{1}{kB^3 \log^{3/2} kB}\right)$, we obtain an upper bound on the expected cover time on the n cycle of $O(k^2 B^5 \log^{5/2} kB \cdot n \log n)$. The result follows by bounding $O(k^2 B^5 \log^{5/2} kB)$ by $\text{poly}(k) \cdot \text{poly}(B)$. □

3 Lower Bound on the Cycle

This section is dedicated to proving Theorem 3. Consider a k-intermittent search X on the cycle C_n and denote by $(p_i)_{i=1}^k$ and $(L_i)_{i=1}^k$ its parameters with $L_i < L_{i+1}$ for all $i \in [k-1]$. We also set $L_{k+1} = n$. Theorem 3 is a direct consequence of the following lemma, as appears in the full version (in short, the biggest multiplicative gap between consecutive L_i is minimized when setting $L_i = n^{\frac{i}{k}}$ and this yields the lower bound).

Lemma 13. There exists a constant $c > 0$ such that, in expectation, for any $i \leq k$, the time needed for X to visit n distinct points is at least

$$c\frac{n}{k} \cdot \sqrt{\frac{L_{i+1}}{L_i}},$$

Proof. Let $i \in [k]$ be fixed throughout the proof. Recall that we need to count time and not the number of moves. We divide time into *phases*, each of length precisely L_{i+1}. We call any jump of length L_j for $j \geq i+1$ a *long jump.* By definition, during a phase, at most one endpoint of a long jump is visited. Let us denote by N_ℓ the number of nodes visited during phase ℓ (some of these nodes may have been previously visited on a phase $\ell' < \ell$). The proof of the following claim appears in the full version.

Claim 14. For every $\ell \in \mathbb{N}$, it holds that $\mathbb{E}(N_\ell) = O\left(k\sqrt{L_i \cdot L_{i+1}}\right)$.

Using Claim 14, we can bound the total number of nodes visited during the first s phases $\mathbb{E}\left(\sum_{\ell=1}^{s} N_\ell\right) \leq s \cdot O\left(k\sqrt{L_i \cdot L_{i+1}}\right)$. Let $s_1 := n \cdot \frac{c}{k \cdot \sqrt{L_i \cdot L_{i+1}}}$ for a small constant c. With this choice for s, the previous bound is less than $n/2$. Using a Markov inequality, we get $\Pr\left(\sum_{\ell=1}^{s_1} N_\ell \geq n\right) < \frac{1}{2}$. Consequently, with probability at least $1/2$, more than s_1 phases are needed in order to visit n distinct nodes. Since each phase lasts (exactly) L_{i+1} time, the total expected time required in order to visit n distinct nodes is at least $s_1 \cdot L_{i+1} = \Omega\left(\frac{n}{k} \cdot \sqrt{\frac{L_{i+1}}{L_i}}\right)$. □

4 Efficient Strategy for Walk or Probe

Our strategy for the *Walk or Probe* problem is simple: instead of probing at every step, we probe every $\sqrt{t_{mix}}$ steps (we omit ceilings for readability). We prove here that this tactic gives the bounds of Theorem 4. In fact, here we essentially prove a bound on the cover time of the Markov chain $(X_{k\sqrt{t_{mix}}})_{k\geq 0}$, where $(X_t)_t$ is the lazy random walk on the connected graph G.

Proof. We first recall some basic results about mixing time. The notion of mixing time we refer to is the total variation mixing time [29, Sect. 4.5]. It is defined as:

$$t_{mix} = \min\left\{t \geq 1 : \max_x \sum_y |P^t(x,y) - \pi(y)| \leq \frac{1}{2}\right\},$$

where $P^t(x,\cdot)$ denotes the law of the random walk started at x after t steps and π is the stationary distribution. Lemmas 4.5 and 4.7 in [2] imply that, for $t \geq 4t_{mix}, P^t(x,y) \geq \frac{\pi(y)}{2}$. Since $\pi(x) = \Delta(x)/2|E|$, where $\Delta(x)$ is the degree of node x, for $t \geq 4t_{mix}$, we have:

$$\Pr(X_t = x) \geq \frac{\Delta(x)}{4|E|}. \tag{12}$$

Let $\mathcal{N}$ be the number of times we probe x between times $4t_{mix}$ and $(4+c)t_{mix}$. Then, by Eq. (8), the probability that x was probed in this time interval equals

$$\frac{\mathbb{E}(\mathcal{N})}{\mathbb{E}(\mathcal{N}|\mathcal{N}\geq 1)}. \tag{13}$$

By Eq. (12) we have:

$$\mathbb{E}(\mathcal{N}) = \sum_{k=0}^{c\sqrt{t_{mix}}} \Pr(X_{4t_{mix}+k\sqrt{t_{mix}}} = x) \geq c\sqrt{t_{mix}}\frac{\Delta(x)}{4|E|}.$$

For the denominator in (13), since the process is markovian, we can shift the times and so this is equal to $\mathbb{E}\left(N_x(c\,t_{mix})|N_x(c\,t_{mix})\geq 1)\right)$ where $N_x(c\,t_{mix})$ is the number of times we probe x before time $c\,t_{mix}$. As the number of returns is maximized whenever we begin at x, this is less than $\mathbb{E}\left(N_x(c\,t_{mix})|X_0 = x\right)$. We next use the following bounds on the probability of returns. For any $x\in G$,

$$\Pr(X_t = x) \leq 5/\sqrt{t} \quad \text{if } t\leq 5n^2 \text{ and } G \text{ is regular}, \tag{14}$$
$$\leq \Delta(x)/\sqrt{t} \quad \text{if } t \leq |E|^2 - 1. \tag{15}$$

The bound for regular graphs is taken from Proposition 6.18 in [2], while the general bound follows from the more elaborate bound in Lemma 3.4 in [32]. Let us write these bounds as

$$\Pr(X_t = x) \leq \frac{\beta_x}{\sqrt{t}}$$

with $\beta_x = 5$ if G is regular and $\beta_x = \Delta(x)$ otherwise. Note that we can use these bounds for $t\leq c\,t_{mix}$, for c small enough, since $t_{mix}\leq 3t_{cov}$ [29, Eq. (10.24)] and, in connected graphs $t_{cov}\leq 2|E|(n-1)\leq 2|E|^2$ [3, Theorem], while in regular graphs we have $t_{cov}\leq 2n^2$ [17, Corollary 6]. Thus, using that $\sum_{k=1}^{t}\frac{1}{\sqrt{k}}\leq 2\sqrt{t}$, we have:

$$\mathbb{E}(N_x(c\,t_{mix})|X_0 = x) = \sum_{k=0}^{c\sqrt{t_{mix}}} \Pr(X_{k\sqrt{t_{mix}}} = x|X_0 = x)$$
$$\leq 1 + \frac{\beta_x}{t_{mix}^{\frac{1}{4}}}\sum_{k=1}^{c\sqrt{t_{mix}}}\frac{1}{\sqrt{k}} \leq 1 + 2\beta_x\sqrt{c}.$$

Hence, the probability that x is probed before time $(4+c)t_{mix}$ is greater than

$$\frac{\Delta(x)}{4|E|}\frac{c\sqrt{t_{mix}}}{1+2\beta_x\sqrt{c}} = \Omega\left(\frac{\sqrt{t_{mix}}}{\gamma}\right),$$

where $\gamma = n$ if G is regular and $\gamma = |E|$ otherwise. By Lemma 8 applied to the Markov chain $(X_{k\sqrt{t_{mix}}})_{k\geq 0}$, the expected number of moves to probe all nodes is $O(\gamma\sqrt{t_{mix}}\log(n))$. Since we probe every $\sqrt{t_{mix}}$ steps with cost C, and a step has a unit cost, the probing cover time is $O(C\gamma\log(n) + \gamma\sqrt{t_{mix}}\log(n))$. This completes the proof of Theorem 4. □

References

1. Adler, M., Räcke, H., Sivadasan, N., Sohler, C., Vöcking, B.: Randomized pursuit-evasion in graphs. Comb. Probab. Comput. **12**(3), 225–244 (2003)
2. Aldous, D., Fill, J.A.: Reversible Markov chains and random walks on graphs. Unfinished monograph, recompiled 2014 (2002). http://www.stat.berkeley.edu/aldous/RWG/book.html
3. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: SFCS. IEEE Computer Society, Washington, D.C., USA (1979)
4. Bartumeus, F., Catalan, J., Fulco, U.L., Lyra, M.L., Viswanathan, G.M.: Optimizing the encounter rate in biological interactions: Lévy versus Brownian strategies. Phys. Rev. Lett. **88**, 097901 (2002)
5. Bartumeus, F.: Lévy processes in animal movement: an evolutionary hypothesis. Fractals **15**(02), 151–162 (2007)
6. Bénichou, O., Loverdo, C., Moreau, M., Voituriez, R.: Intermittent search strategies. Rev. Mod. Phy. **83**(1), 81 (2011)
7. Benjamini, I., Kozma, G., Wormald, N.: The mixing time of the giant component of a random graph. Random Struct. Algorithms **45**(3), 383–407 (2014)
8. Berg, O.G., Winter, R.B., Von Hippel, P.H.: Diffusion-driven mechanisms of protein translocation on nucleic acids. 1. Models and theory. Biochemistry **20**(24), 6929–6948 (1981)
9. Boyer, D., et al.: Scale-free foraging by primates emerges from their interaction with a complex environment. Proc. R. Soc. Lond. B Biol. Sci. **273**(1595), 1743–1750 (2006)
10. Chupeau, M., Benichou, O., Voituriez, R.: Cover times of random searches. Nat. Phy. **11**(10), 844 (2015)
11. Cooper, C., Frieze, A.: The cover time of the Giant component of a random graph. Random Struct. Algorithms **32**(4), 401–439 (2008)
12. Coppersmith, D., Feige, U., Shearer, J.B.: Random walks on regular and irregular graphs. SIAM J. Discret. Math. **9**(2), 301–308 (1996)
13. Coppey, M., Bnichou, O., Voituriez, R., Moreau, M.: Kinetics of target site localization of a protein on DNA: a stochastic approach. Biophys. J. **87**(3), 1640–1649 (2004)
14. Czyzowicz, J., Gçsieniec, L., Georgiou, K., Kranakis, E., MacQuarrie, F.: The Beachcombers' problem: walking and searching with mobile robots. Theor. Comput. Sci. **608**(Part 3), 201–218 (2015). Structural Information and Communication Complexity
15. Edwards, A.M., et al.: Revisiting levy flight search patterns of wandering albatrosses, bumblebees and deer. Nature **449**(7165), 1044 (2007)
16. Einstein, A.: Investigations on the theory of the Brownian movement. Annal. Physik **34**, 591–592 (1911)
17. Feige, U.: Collecting coupons on trees, and the analysis of random walks. Technical report (1994)
18. Feige, U.: A tight lower bound on the cover time for random walks on graphs. Random Struct. Algorithms **6**(4), 433–438 (1995)
19. Fonio, E., Heyman, Y., Boczkowski, L., Gelblum, A., Kosowski, A., Korman, A., Feinerman, O.: A locally-blazed ant trail achieves efficient collective navigation despite limited information. eLife **5** (2016)

20. Friedman, J.: A proof of Alon's second eigenvalue conjecture and related problems. CoRR, cs.DM/0405020 (2004)
21. Harris, T.H., et al.: Generalized Lévy walks and the role of chemokines in migration of effector CD8(+) T cells. Nature **486**, 545 (2012)
22. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: PODC. ACM (1990)
23. Kanade, V., Mallmann-Trenn, F., Sauerwald, T.: On coalescence time in graphs-when is coalescing as fast as meeting? CoRR, abs/1611.02460 (2016)
24. Kempe, D., Kleinberg, J.M., Demers, A.J.: Spatial gossip and resource location protocols. J. ACM **51**(6), 943–967 (2004)
25. Klafter, J., Zumofen, G.: Lévy statistics in a Hamiltonian system. Phys. Rev. E **49**, 4873 (1994)
26. Klafter, J., Shlesinger, M.F., Zumofen, G.: Beyond Brownian motion. Phys. Today **49**(2), 33 (1996)
27. Kleinberg, J.M.: The small-world phenomenon: an algorithmic perspective. In: STOC (2000)
28. Lawler, G., Limic, V.: Random Walk: A Modern Introduction. Cambridge University Press, Cambridge (2010)
29. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing. American Mathematical Society, Providence (2008)
30. Lomholt, M.A., et al.: Lévy strategies in intermittent search processes are advantageous. Proc. Natl. Acad. Sci. **105**(32), 11055–11059 (2008)
31. Loverdo, C.: Optimal search strategies and intermittent random walk: from restriction enzymes to the albatross flight, December 2009
32. Lyons, R.: Asymptotic enumeration of spanning trees. Comb. Probab. Comput. **14**(4), 491–592 (2005)
33. Oshanin, G., et al.: Intermittent random walks for an optimal search strategy: one-dimensional case. J. Phy. Condens. Matter **19**(6), 065142 (2007)
34. Pearson, K.: The problem of the random walk. Nature **72**, 1905 (1865)
35. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 17:1–17:24 (2008)
36. Reynolds, G.: Navigating our world like birds and bees. The New York Times (2014)
37. Rhee, I., Shin, M., Hong, S., Lee, K., Chong, S.: On the Levy-walk nature of human mobility. In: IEEE INFOCOM 2008 (2008)
38. Travis, J.: Do wandering albatrosses care about math? Science **318**(5851), 742–743 (2007)
39. Viswanathan, G.M., et al.: Levy flight search patterns of wandering albatrosses. Nature **381**(6581), 413 (1996)
40. Viswanathan, G.M., et al.: Optimizing the success of random searches. Nature **401**(6756), 911 (1999)

Tight Kernels for Covering and Hitting: Point Hyperplane Cover and Polynomial Point Hitting Set

Jean-Daniel Boissonnat[1], Kunal Dutta[1], Arijit Ghosh[2](✉), and Sudeshna Kolay[3]

[1] Université Côte d'Azur, Inria, Sophia Antipolis, France
{Jean-Daniel.Boissonnat,Kunal.Dutta}@inria.fr
[2] Indian Statistical Institute, Kolkata, India
agosh@mpi-inf.mpg.de
[3] Eindhoven University of Technology, Eindhoven, Netherlands
s.kolay@tue.nl

Abstract. The Point Hyperplane Cover problem in $\mathbb{R}^d$ takes as input a set of n points in $\mathbb{R}^d$ and a positive integer k. The objective is to cover all the given points with a set of at most k hyperplanes. The D-Polynomial Points Hitting Set (D-Polynomial Points HS) problem in $\mathbb{R}^d$ takes as input a family $\mathcal{F}$ of D-degree polynomials from a vector space $\mathcal{R}$ in $\mathbb{R}^d$, and determines whether there is a set of at most k points in $\mathbb{R}^d$ that hit all the polynomials in $\mathcal{F}$. For both problems, we exhibit tight kernels where k is the parameter.

1 Introduction

A set system is a tuple $(U, \mathcal{F})$ where U is a universe of n elements and $\mathcal{F}$ is a family of m subsets of U. A set system is also referred to as a *hypergraph*, with the elements in the universe U named as *vertices* and the subsets in $\mathcal{F}$ named as *hyperedges*. A hyperedge is said to *cover* a vertex if the vertex belongs to the hyperedge. Similarly a subfamily $\mathcal{F}'$ of hyperedges is said to cover a subset V of vertices if for each vertex $v \in V$ there is a hyperedge $h \in \mathcal{F}'$ such that h covers v. A vertex is said to *hit* a hyperedge if the vertex belongs to the hyperedge, and a subset V of vertices is said to hit a subfamily $\mathcal{F}'$ of hyperedges if for each hyperedge $h \in \mathcal{F}'$ there is a vertex $v \in V$ that belongs to h.

The Set Cover and Hitting Set problems are two of the most well-studied problems in computer science. For the Set Cover problem, the input is a set system $(U, \mathcal{F})$ and a positive integer k. The objective is to determine whether there is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with at most k subsets, such that $\mathcal{F}'$ covers all the elements in U. Such a family $\mathcal{F}'$ is referred to as a *solution family* or a *covering family*. The Hitting Set problem can be thought of as a dual problem. Here, the input is the same as in Set Cover. However, now the objective is to determine whether there is a subset $S \subseteq U$ of size at most k, such that for each hyperedge $h \in \mathcal{F}$, $h \cap S \neq \emptyset$. Such a set S is referred to as a *solution set*.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 187–200, 2018.
https://doi.org/10.1007/978-3-319-77404-6_15

These problems are part of the original 21 NP-complete problems posed by Karp [15]. However, the numerous applications for these problems inspired researchers to design algorithms to find solutions with reasonable efficiency, for different measures of efficiency. For Set Cover, the best approximation factor is $\mathcal{O}(\log n)$ [20]. It was shown in [9] that $\log n$ is the best possible approximation factor unless $P = NP$. Since Hitting Set is just a reformulation of the Set Cover problem, the same approximation factors hold. The d-Hitting Set problem, where the size of each subset in $\mathcal{F}$ is exactly d, is known to be APX-hard [2], consequent to results obtained for the special case of the Vertex Cover problem where $d = 2$.

Set Cover and Hitting Set have been studied in parameterized complexity. In parameterized complexity, we say that a problem is fixed parameter tractable (FPT) with respect to a parameter k, if there is an algorithm that takes an instance of size n of the problem, and solves the problem in $f(k).n^{\mathcal{O}(1)}$ time, where f is a computable function. For a brief introduction to parameterized complexity please refer to the Preliminaries. For further details please refer to [7, 11, 12]. The d-Hitting Set problem, parameterized by the solution size k, is known to be FPT, with a tight $\mathcal{O}(k^d)$-sized kernel [8] under standard complexity theoretic assumptions. In this paper, unless otherwise mentioned, all variants of Hitting Set and Set Cover are parameterized by k. The Set Cover and the general Hitting Set problems are W[2]-hard, and are not expected to be FPT.

Interestingly, the instances of many real world applications of these two notoriously hard problems have inherent structure in them. With the hope of designing efficient algorithms for such instances by exploiting their structural information, numerous variants of Set Cover and Hitting Set have been studied. A very natural extension in this field of study is to assume geometric structure on the instances. In recent years, there has been a lot of attention to study geometric variants of both the problems.

The Point Line Cover is an example of a geometric variant of Set Cover, where the universe is a set of points in $\mathbb{R}^2$ and the hyperedges are the maximal sets of collinear points in the input. Point Line Cover is known to be FPT [17]. Kratsch et al. showed in [16] that the problem has a tight polynomial kernel with $\mathcal{O}(k^2)$ points. In [1], several generalizations of the Point Line Cover problem were studied - a universe is a set of points in a Euclidean space and the family of hyperedges are geometric structures like hyperplanes, spheres, curves, etc. Geometric variants of Set Cover have been studied in [3, 4].

The results in [16] also imply parameterized results for Line Point Hitting Set, where the universe is a set of lines in $\mathbb{R}^2$ and the objective is to find at most k points in $\mathbb{R}^2$ to hit the universe of lines. This problem is FPT and has a tight kernel with $\mathcal{O}(k^2)$ lines. Other geometric variants of the Hitting Set problem has been studied in parameterized complexity [10, 13, 14]. Bringmann et al. [6] studied the problem for set systems with bounded VC dimensions. They showed that there are set systems with VC dimension as low as 2, where both the Hitting Set, and consequently the Set Cover problem are W[1]-hard. This gives an interesting dichotomy, since they also show that when the VC dimension of the set system is 1, then the Hitting Set problem is in P.

In this paper, we consider two parameterized variants.

POINT HYPERPLANE COVER IN $\mathbb{R}^d$ **Parameter:** k
Input: A set $\mathcal{P}$ of n points in $\mathbb{R}^d$, a positive integer k.
Question: Is there a family of at most k hyperplanes in $\mathbb{R}^d$ that cover all the points in $\mathcal{P}$?

We also study the PROJECTIVE POINT HYPERPLANE COVER problem, where the family of hyperplanes allowed to cover the input set of points must pass through the origin in $\mathbb{R}^d$, and we are not allowed to include the origin in the input set of points. Note that this problem is equivalent to that of covering points on a sphere with the great circles (radius is equal to the radius of the sphere) of the sphere, which has many applications in computational geometry.

D-POLYNOMIAL POINT HS[a] IN $\mathbb{R}^d$ **Parameter:** k
Input: A set $\mathcal{F}$ of n D-degree polynomials from a specified vector space $\mathcal{R}$ of D-degree polynomials in $\mathbb{R}^d$, a positive integer k.
Question: Is there a set S of at most k points in $\mathbb{R}^d$ such that for each polynomial $f \in \mathcal{F}$, there is a point $p \in S$ with $f(p) = 0$?

[a] HS is a shorthand for hitting set.

Please refer to the Preliminaries for the definition of D-degree polynomials. Both problems are NP-hard, because of the NP-hardness of POINT LINE COVER [19]. Parameterized by k, we study the parameterized complexity of these problems.

Our results. Extending the results of Kratsch et al. [16], we show that POINT HYPERPLANE COVER and PROJECTIVE POINT HYPERPLANE COVER in $\mathbb{R}^d$ have tight polynomial kernels with $\mathcal{O}(k^d)$ and $\mathcal{O}(k^{d-1})$ points, respectively. These results are presented in Sect. 3. The highlight of this proof is, given any positive integer n, the construction of a set $\mathcal{P}$ of n points in $\mathbb{R}^d$ in general position such that the family of hyperplanes, defined by any d points from $\mathcal{P}$, do not have too many hyperplanes intersecting at a point outside $\mathcal{P}$. This is crucial for a many-one reduction from d-HITTING SET, that results in lower bounds on the size as well as the number of points in a kernel under complexity theoretic assumptions. The construction is similar to that in [16] in spirit, but requires more geometric insight since we are working in higher dimensions now.

Note that the results of POINT HYPERPLANE COVER also imply that the dual problem, HYPERPLANE POINT HS in $\mathbb{R}^d$, also has a tight kernel with $\Omega(k^d)$ hyperplanes. Similarly, we can show that PROJECTIVE HYPERPLANE POINT COVER has a tight kernel with $\Omega(k^{d-1})$ hyperplanes.

Our main contribution is to show tight polynomial bounds for kernel sizes for D-POLYNOMIAL POINT HS in $\mathbb{R}^d$, for a large family of vector spaces $\mathcal{R}$ of D-degree polynomials. For more details on the characterization of $\mathcal{R}$, please see the Preliminaries. The vector space of hyperplanes, spheres and ellipses are among natural vector spaces of polynomials that are covered by this characterization. Therefore, our techniques provide a general framework for proving tight kernels for covering problems, as one can get a tight bound for many families directly. This result is given in Sect. 4.

Our proof strategy is to use the Veronese mapping [18] to transform the space of points and polynomials to a higher dimensional space, where the polynomials transform into hyperplanes and point-polynomial incidences are preserved. The upper bound on the kernel size comes directly from the Veronese mapping. For the lower bound, we show that points in general position with respect to polynomials transform to an equal-sized set of points in general position with respect to hyperplanes in the image space. Using this fact, we construct hard instances of D-Polynomial Point HS. In [5], the Veronese mapping was used to give upper bounds on kernel sizes. In this paper, we also utilize the map to exhibit kernel lower bounds.

2 Preliminaries

Multivariate Polynomials. Given a set $\{X_1, X_2, \ldots, X_d\}$ of variables a real multivariate polynomial on these variables is of the form $P(X_1, \ldots, X_d) = \sum_{i_1,i_2,\ldots,i_d} a_{i_1 i_2 \ldots i_d} \prod_{j\in[d]} X_j^{i_j}$ where $[d] = \{1, \ldots, d\}$ and $a_{i_1 i_2 \ldots i_d} \in \mathbb{R}$. The set of all real multivariate polynomials in the variables $X_1, \ldots, X_d$ will be denoted by $\mathbb{R}[X_1, X_2, \ldots, X_d]$. The degree of such a polynomial $P(X_1, \ldots, X_d)$ is defined as $\deg(P) := \mathsf{max}\{i_1 + i_2 + \ldots + i_d \mid a_{i_1 i_2 \ldots i_d} \neq 0\}$. A polynomial is said to be a D-degree polynomial if its degree is D.

In this paper, we are interested in the set/subsets of polynomials whose degree is bounded by D, for some $D \in \mathbb{N}$. In this context we define $\mathrm{Poly}_D[X_1, \ldots, X_d] := \{f(X_1, \ldots, X_d) \in \mathbb{R}[X_1, \ldots, X_d] \mid \deg(f) \leq D\}$. Observe that the vector space $\mathrm{Poly}_D[X_1, \ldots, X_d]$ over $\mathbb{R}$ has the *monomials* $\left\{X_1^{i_1} \ldots X_d^{i_d} \mid 0 \leq \sum_{j=1}^{d} i_j \leq D\right\}$ as a basis and $\left|\left\{X_1^{i_1} \ldots X_d^{i_d} \mid 0 \leq \sum_{j=1}^{d} i_j \leq D\right\}\right| = \binom{D+d}{D}$. For ease of notation, we define the vector $X = (X_1, \ldots, X_d)$.

Given a polynomial f and a point p, the point *hits* the polynomial if $f(p) = 0$. In the same situation, the polynomial is said to *cover* the point.

General position in Geometry. An i-flat in $\mathbb{R}^d$ is the affine hull of $i+1$ affinely independent points. The dimension of a (possibly infinite) set of points $\mathcal{P}$, denoted as $\mathsf{dim}(\mathcal{P})$, is the minimum i such that the entire set $\mathcal{P}$ is contained in an i-flat of $\mathbb{R}^d$ [17]. We use the term hyperplanes interchangeably for $(d-1)$-flats. A set $\mathcal{P}$ of points in $\mathbb{R}^d$ is said to be in general position with respect to hyperplanes, if for each i-flat, $i \leq d-1$, in $\mathbb{R}^d$ there are at most $i+1$ points from $\mathcal{P}$ lying on the i-flat.

Consider, for $i \leq d-1$, a family $\mathcal{F}$ of i-flats such that there is a point p that belongs to all the i-flats in $\mathcal{F}$. Then a set $\mathcal{P} \in \mathbb{R}^d \setminus \{p\}$ of points is said to be in general position with respect to $\mathcal{F}$ if each i-flat contains at most i points from $\mathcal{P}$ lying on the i-flat. This is called general position in projective geometry.

Similarly, we can define the notion of general position (resp. *projective* general position) with respect to multivariate polynomials. Let $\mathcal{R}$ be a vector space of $\text{Poly}_D[X]$, defined by a basis $\{f_1(X), \ldots, f_b(X), 1\}$ (resp. by a basis $\{f_1(X), \ldots, f_b(X)\}$ with $\deg(f_i) > 0$). A subset of points is said to be in general position (resp. projective general position) with respect to the vector space $\mathcal{R}$ of polynomials if no more than b points (resp. $b-1$ points) from the subset satisfy any equation of the form $f(X) := \sum_{i=1}^{b} \lambda_i f_i(X) + \lambda_{b+1} = 0$ ($f(X) := \sum_{i=1}^{b} \lambda_i f_i(X) = 0$), where all the $\lambda_j \in \mathbb{R}$ and not all the λ_j's can be zero simultaneously.

Definition 1. *Given a rational number $\alpha > 0$, a vector space $\mathcal{R}$ of polynomials in $\mathbb{R}^d$ is said to be α-good if for any positive integers b, m the following conditions hold:*

1. *In $\mathcal{O}(1)$ time we can compute a set of b points in $\mathbb{R}^d$ such that the set is in general position with respect to $\mathcal{R}$.*
2. *Given a d-dimensional $m \times \cdots \times m$ grid in $\mathbb{R}^d$, each polynomial in $\mathcal{R}$ contains at most $m^{d-\alpha}$ vertices of the grid.*

Hyperplanes, spheres, ellipses and many other natural vector spaces of polynomials can be described as α-good vector spaces.

Veronese mapping. In this paper, one of our strategies for generalizing our results is to convert D-Polynomial Point HS in $\mathbb{R}^d$ to Hyperplane Point HS in $\mathbb{R}^b$ by using a variant of Veronese mapping [18] from $\mathbb{R}^d \rightarrow \mathbb{R}^b$. The Veronese mapping of a vector space $\mathcal{R}$ of D-degree polynomials, with a basis $\{f_1(X), \ldots, f_b(X), 1\}$ (also with a basis $\{f_1(X), \ldots, f_b(X)\}$ where $\deg(f_i) > 0$), is as follows- $\Phi_{\mathcal{R}} : \mathbb{R}^d \rightarrow \mathbb{R}^b$, where $\Phi_{\mathcal{R}}(X) = (f_1(X), \ldots, f_b(X))$ where $X = (X_1, \ldots, X_d)$. Observe that if $p = (p_1, \ldots, p_d)$ satisfies the equation $f(X) := \sum_{i=1}^{b} \lambda_i f_i(X) + \lambda_{b+1} = 0$ (resp. the equation $f(X) := \sum_{i=1}^{b} \lambda_i f_i(X) = 0$) then $\Phi_{\mathcal{R}}(p)$ will also satisfy the linear equation $\sum_{j=1}^{b} \lambda_j Z_j + \lambda_{b+1} = 0$ (or the equation $\sum_{j=1}^{b} \lambda_j Z_j = 0$), on the variable vector $Z = (Z_1, \ldots, Z_b)$. In other words, for any set of points $\mathcal{P}$ in $\mathbb{R}^d$ and $\mathcal{F}$, the incidences between $\mathcal{P}$ and $\mathcal{R}$ and incidences between $\Phi_{\mathcal{R}}(\mathcal{P})$ and hyperplanes in $\mathbb{R}^b$ (or the hyperplanes passing through the origin in $\mathbb{R}^b$) are preserved under the mapping $\Phi_{\mathcal{R}}$. Also, observe that there is a bijection between polynomials in $\mathcal{R}$ and hyperplanes in $\mathbb{R}^b$ (resp. hyperplanes passing through the origin in $\mathbb{R}^b$). This transformation from polynomials to hyperplanes is also referred to as *linearization*.

Parameterized Complexity. The instance of a parameterized problem or language is a pair containing the actual problem instance of size n and a positive integer called a parameter, usually represented as k. The problem is said to be in FPT if there exists an algorithm that solves the problem in $f(k) \cdot n^{\mathcal{O}(1)}$ time, where f is a computable function. The problem is said to admit a $g(k)$-sized kernel, if there exists a polynomial time algorithm that converts the actual instance to a reduced instance of size $g(k)$, while preserving the answer. When g is a polynomial function, then the problem is said to admit a polynomial kernel. A reduction rule is a polynomial time procedure that changes a given instance I_1 of a problem Π to another instance I_2 of the same problem Π. We say that the reduction rule is *safe* when I_1 is a YES instance of Π if and only if I_2 is a YES instance. Readers are requested to refer to [7] for more details on Parameterized Complexity.

Lower bounds in Parameterized Algorithms. There are several methods of showing lower bounds in parameterized complexity under standard assumptions in complexity theory. In this paper we require a lower bound technique given by Dell and Melkebeek [8]. This technique links kernelization to oracle protocols.

Definition 2. *Given a language L, an oracle communication protocol for L is a two-player communication protocol. The first player gets an input x and can only execute computations taking time polynomial in $|x|$. The second player is computationally unbounded, but does not know x. At the end of the protocol, the first player has to decide correctly whether $x \in L$. The cost of the protocol is the number of bits of communication from the first player to the second player.*

Proposition 3 [8]**.** *Let $d \geq 2$ be an integer, and ϵ be a positive real number. If* co-NP $\not\subseteq$ NP*/poly, then there is no protocol of cost $\mathcal{O}(n^{d-\epsilon})$ to decide whether a d-uniform hypergraph on n vertices has a d-hitting set of at most k vertices, even when the first player is co-nondeterministic.*

As noted in [8], this implies that for any $d \geq 2$ and any positive real number ϵ, if co-NP $\not\subseteq$ NP/poly, then there is no kernel of size $\mathcal{O}(k^{d-\epsilon})$ for d-HITTING SET. In general, a lower bound for oracle communication protocols for a parameterized language L gives a lower bound for kernelization for L.

Kernels: size vs number of elements. In the literature, a lower bound on the kernel implies a lower bound on the size in bits of the kernel, but not necessarily on the number of input elements in the kernel. Kratsch et al. [16] were one of the first to study lower bounds in terms of the number of input elements in the kernel. They used the results of Dell and Melkebeek [8] along with results in two dimensional geometry to build a new technique to show lower bounds for the number of input elements in a kernel for a problem. In this paper, we have adhered to the general convention by saying that a kernel has a lower bound on its size if it has a lower bound on its representation in bits, while explicitly mentioning the cases where the kernel has a lower bound on the number of input elements.

3 Kernelization Lower Bound for POINT HYPERPLANE COVER

In this section, we show that POINT HYPERPLANE COVER in $\mathbb{R}^d$ has a tight kernel of size $\Theta(k^d)$. The results in [17] imply that POINT HYPERPLANE COVER in $\mathbb{R}^d$ has a kernel of size $\mathcal{O}(k^d)$. We show that the problem cannot have a kernel of size $\mathcal{O}(k^{d-\epsilon})$ if co-NP $\nsubseteq$ NP/poly. We show this by the standard technique of polynomial parameter transformation. For a fixed d, we reduce the d-Hitting Set problem to the problem of Point Hyperplane Cover in $\mathbb{R}^d$. We first state the folklore equivalence between Point Hyperplane Cover in $\mathbb{R}^d$ and Hyperplane Point HS in $\mathbb{R}^d$.

Lemma 4. Point Hyperplane Cover *in* $\mathbb{R}^d$ *and* Hyperplane Point *HS in* $\mathbb{R}^d$ *are equivalent problems.*

From now on, we will be showing lower bounds for Hyperplane Point HS. The proof strategy is the same as that in [16]. For this, we construct for each positive integer n and each d, a set of n points in $\mathbb{R}^d$ with some special properties. This construction is more involved than in the case of Point Line Cover.

Lemma 5. *For every* $n \in \mathbb{Z}^+$*, there is a* poly(n) *time algorithm to construct a set* $\mathcal{P}$ *of* n *points in* $\mathbb{R}^d$ *that have the following properties:*

(1) *The points are in general position.*
(2) Let $\mathcal{H}$ *be the family of hyperplanes defined by all sets of* d *points from* $\mathcal{P}$*. The hyperplanes in the family* $\mathcal{H}$ *are in general position, i.e., given* r *hyperplanes* $H_1, \ldots, H_r$ *in* $\mathcal{H}$ *with* $r \leq d$ *the dimension of the affine space* $\cap_{i=1}^r H_r$ *is* $d-r$*.*
(3) For any point p *in* $\mathbb{R}^d \backslash \mathcal{P}$*, there are at most* d *hyperplanes in* $\mathcal{H}$ *that contain* p*.*

Proof. The set $\mathcal{P}$ is built inductively. When $n = d$, it is the base case and the construction follows trivially by taking any n points in general position. There is exactly one hyperplane that is defined by this set of d points, therefore all required conditions are met. Now, assume that for $d \leq t < n$, we have constructed a point set $\mathcal{P}_t$ that satisfies the above conditions. As in [16], our goal will be to extend the point set $\mathcal{P}_t$ by one point. We will show that points forbidden to be added to the set $\mathcal{P}_t$ will lie on a bounded number of hyperplanes and we will call these hyperplanes *forbidden hyperplanes.* Observe that the number of forbidden hyperplanes arising due to condition (1) is $O(t^d)$. A forbidden hyperplane due to condition (2) is defined by the intersection space of a set of at most d hyperplanes and a set of at most $d-1$ points from $\mathcal{P}_t$. Therefore, the bound on the number of forbidden hyperplanes arising from condition (2) is $O(t^{d^2+d-1})$.

Unlike the case when $d = 2$, it is harder to bound the number of forbidden hyperplanes due to condition (3). Let $q \in \mathbb{R}^d$ be a point where the point set $\mathcal{P}' = \mathcal{P}_t \cup \{q\}$ satisfies conditions (1) and (2), but not condition (3). We will call such a point q a *forbidden point.* Let $\mathcal{H}'$ be the family of hyperplanes defined by each set of d points from $\mathcal{P}'$. Let $H_1, \ldots, H_{d+1}$ be a set of $d+1$ hyperplanes in $\mathcal{H}'$ such that they intersect at point s with $s \in \mathbb{R}^d \setminus \mathcal{P}'$. Observe that since the point set $\mathcal{P}_t$ satisfied all the three conditions, q will lie on at least 1 hyperplane from the family $\{H_1, \ldots, H_{d+1}\}$. Suppose q was contained in at least d hyperplanes from the family, then $q = s$ as $\mathcal{P}'$ satisfies condition (2). Therefore, it must be the case that q lies in at least 1 and at most $d-1$ hyperplanes from the family $\{H_1, \ldots, H_{d+1}\}$. Without loss of generality, assume that q lies on the hyperplanes $\{H_1, \ldots, H_r\}$. Let A_{r-1} denote the $(r-1)$ dimensional affine plane $\cap_{i=r+1}^{d+1} H_i$. For $j \in [r]$, let the hyperplanes H_j be generated by the set $\{q, p_1^j, \ldots, p_{d-1}^j\} \subset \mathcal{P}'$. The point s also belongs to A_{r-1}. Since we are interested in understanding where the forbidden point q can lie, we try to understand the inverse problem where A_{r-1}, s, and points p_ℓ^j (for all $\ell \in [d-1]$) are fixed and q is the variable point such that $\cap_{i=1}^{d+1} H_i = s \in \mathbb{R}^d \setminus \mathcal{P}'$. Using elementary Euclidean geometry, we get that at least $d-r+1$ coordinates of s are fixed when A_{r-1} gets fixed. We know that q lies on a $d-r+1$ dimensional affine plane passing through s. Since $d-r+1$ coordinates of s are fixed by A_{r-1}, the slope of the affine plane depends only on A_{r-1} and the points p_ℓ^j, $j \in [r]$ and $\ell \in [d-1]$. This implies that as we vary s on A_{r-1} we will span a hyperplane which only depends on A_{r-1} and the points p_ℓ^j, $j \in [r]$ and $\ell \in [d-1]$. Therefore, once the hyperplanes H_{r+1} till H_{d+1} and the point set $\{p_t^j | j \in [r], \ell \in [d-1]\}$ are fixed, the point q will lie on a unique hyperplane. This implies that the number of forbidden hyperplanes due to condition (3) is bounded by $O(t^{d^2+d-1})$.

As we have an upper bound on the number of forbidden hyperplanes, we can now use the trick of Kratsch et al. to generate points satisfying conditions (1) to (3) [16, Lemma 2.4]. In our case, we take a d-dimensional $m \times \cdots \times m$ grid with $m = n^{d^2+d}$. Observe that the number of points from this d-dimensional grid that can lie on any hyperplane is bounded by m^{d-1}.

Finally, we are ready to prove the following Theorem.

Theorem 6. Hyperplane Point HS *in* $\mathbb{R}^d$ *cannot have a kernel of size* $\mathcal{O}(k^{d-\epsilon})$ *if* co-NP $\nsubseteq$ NP*/poly.*

Proof. We give a reduction from d-Hitting Set. Let $(U, \mathcal{F}, k)$ be an instance of d-Hitting Set. First we reduce this instance to the following instance $(U', \mathcal{F}', dk)$ where:

1. For each $v \in U$ we make d copies $\{v^1, v^2, \ldots, v^d\}$. We also refer to the set $\{v^1, v^2, \ldots, v^d\}$ as the row of v.
2. $U' = U_1 \uplus U_2 \uplus \ldots \uplus U_d$ such that for each $i \in \{1, \ldots, d\}$ $U_i = \{v^i | v \in U\}$.
3. $\mathcal{F} \subset \mathcal{F}'$.
4. Assume that there is an arbitrary ordering on the vertices of $U = \{v_1, v_2, \ldots, v_n\}$. For each $f = \{v_{j_1}, v_{j_2}, \ldots, v_{j_d}\} \in \mathcal{F}$, and for each $i_1, i_2, \ldots, i_d \in \{1, \ldots, d\}$, we create a subset $f_{i_1,i_2,\ldots,i_d} = \{v_{j_1}^{i_1}, v_{j_2}^{i_2}, \ldots, v_{j_d}^{i_d}\}$. We put $f_{i_1,i_2,\ldots,i_d}$ in the set $\mathcal{F}'$.
5. For clarity of arguments in what follows, we give some more definitions. For each $f = \{v_{j_1}, v_{j_2}, \ldots, v_{j_d}\} \in \mathcal{F}$, the subfamily of hyperedges $\mathcal{F}_{v_{j_1},v_{j_2},\ldots,v_{j_d}} = \{f_{i_1,i_2,\ldots,i_d}\} = \{v_{j_1}^{i_1}, v_{j_2}^{i_2}, \ldots, v_{j_d}^{i_d}\} | i_1, i_2, \ldots, i_d \in \{1, \ldots, d\}\}$ is called a subsystem of $\{v_{j_1}, v_{j_2}, \ldots, v_{j_d}\}$. Also, $\mathcal{F}_{v_{j_1},v_{j_2},\ldots,v_{j_d}}$ is called a subsystem of f, for all hyperedges $f \in \mathcal{F}_{v_{j_1},v_{j_2},\ldots,v_{j_d}}$. It follows from the previous definition that a row in a subsystem corresponds to the d copies of a vertex participating in the subsystem.

Claim 1. *$(U, \mathcal{F}, k)$ is a* Yes *instance of d-*Hitting Set *if and only if $(U', \mathcal{F}, dk)$ is a* Yes *instance of d-*Hitting Set*.*

Next, we give a reduction from the instance $(U', \mathcal{F}, dk)$ of d-Hitting Set to a instance of Hyperplane Point HS. The correctness of this reduction shows that there is a polynomial time reduction from d-Hitting Set to Hyperplane Point HS such that the parameter transformation is linear.

We construct the following instance of Hyperplane Point HS:

1. Using Lemma 5, we construct a set $\mathcal{P}$ of dn points, same as the number of elements in the universe U'. We arbitrarily assign each element of U' to a unique point in $\mathcal{P}$.
2. For a hyperedge $f \in \mathcal{F}'$, let H_f be the hyperplane defined by the d points contained in f. The set $\mathcal{H}$ is the family of such hyperplanes.

Claim 2. *$(U, \mathcal{F}, k)$ is a* Yes *instance of d-*Hitting Set *if and only if $(\mathcal{H}, dk)$ is a* Yes *instance of* Hyperplane Point *HS in $\mathbb{R}^d$.*

To prove this claim, we need the following claim regarding a solution set with minimum number of points outside $\mathcal{P}$.

Claim 3. *Let Q be a minimum sized set of points that covers all the hyperplanes in $\mathcal{H}$. Also, assume that Q has the minimum possible points in $Q \setminus \mathcal{P}$. Moreover, let $q \in Q \setminus \mathcal{P}$ that covers the minimum number of hyperedges uniquely. We assume that there is no other set Q' of the same size as Q, with $|Q' \setminus \mathcal{P}| = |Q \setminus \mathcal{P}|$ and with a $q' \in Q' \setminus \mathcal{P}$ that covers strictly less number of hyperedges uniquely in Q' than q does in Q. Then for any element $v \in U' \setminus Q$, at most $d - 1$ hyperedges containing v can have no intersection with Q.*

Proof. Firstly, by the condition of minimality on Q, each point in $Q \setminus \mathcal{P}$ must uniquely cover at least 2 hyperplanes in $\mathcal{H}$. Otherwise we could find a equal-sized solution Q' where $|Q' \setminus \mathcal{P}| < |Q \setminus \mathcal{P}|$, which is a contradiction.

Suppose that there is a vertex $v \in U' \setminus Q$ such that at least a family $\mathcal{H}'$ of d hyperedges in $\mathcal{H}$ containing v can have no intersection with Q. These d hyperplanes are covered by a set Q' of points that are in $Q \setminus \mathcal{P}$. Suppose $Q' = \{u_1, \ldots, u_\ell\}$ such that the for each $j \in \{1, \ldots, \ell\}$, u_j uniquely covers c_j hyperplanes of $\mathcal{H}'$. By definition, $\Sigma_j c_j = d$. By the minimality condition of Q and property (2) of Lemma 5, each such point in $Q' \subseteq Q \setminus \mathcal{P}$ uniquely covers between 2 to d hyperplanes of $\mathcal{H}$. Thus, for each $j \in \{1, 2, \ldots, \ell\}$ the vertex u_j covers at most $d - c_j$ hyperplanes not in $\mathcal{H}'$. We call the family of all hyperplanes covered by vertices of Q' as $\mathcal{H}''$. This family has at most $d(d-1)+d$ hyperplanes. We construct the following set $\hat{Q}$:

- All points of $Q \setminus Q'$ are included in $\hat{Q}$. The point v is also included.
- For each $j \in \{1, \ldots, \ell\}$, let $\mathcal{H}_j$ be the subfamily of at most $d - c_j$ hyperplanes that are uniquely covered by the vertex u_j and which are not in $\mathcal{H}'$. Starting from $j = 1$, we build a subfamily $\mathcal{H}'_j$ and find a point u'_j corresponding to u_j. First all the hyperplanes in $\mathcal{H}_j$ are added to $\mathcal{H}'_j$. Then, iterating a variable t

from $j+1$ to ℓ, we add the hyperplanes in $\mathcal{H}_t$ till there are d hyperplanes or all hyperplanes in $\bigcup_{t \geq j} \mathcal{H}_t$ have been added. Take a point in the intersection of $\mathcal{H}'_j$ and name that point u'_j. We show that the last nonempty subfamily $\mathcal{H}_t$ must be for $t < \ell$. Suppose not. Then by definition, when we consider the last point u_ℓ, the number of hyperplanes in $\mathcal{H}_\ell$ that are not yet covered by $\{u'_1, \ldots, u'_{\ell-1}\}$ are at most $d - c_\ell - \Sigma_{j<\ell} c_j = d - \Sigma_{j \leq \ell} c_j = 0$. Therefore, $\{c'_1, c'_2, \ldots, c_{\ell-1'}\}$ cover all the hyperplanes in $\mathcal{H}'' \setminus \mathcal{H}'$. By definition of $\mathcal{H}'$, the set $\{v, c'_1, c'_2, \ldots, c_{\ell-1'}\}$ covers all the hyperplanes in $\mathcal{H}''$.

By definition the size of $\hat{Q}$ is at most that of Q. However, the number of vertices in $\hat{Q} \setminus \mathcal{P}$ is strictly less than the number of vertices in $Q \setminus \mathcal{P}$. This is a contradiction to the definition of Q.

Hence, we have proven the claim.

The proofs of the other claims can be found in the full version of the paper. Due to Claim 2, we show that there is a linear parameter transformation from d-Hitting Set to Hyperplane Point HS in $\mathbb{R}^d$. This implies that Hyperplane Point HS in $\mathbb{R}^d$ cannot have a kernel of size $\mathcal{O}(k^{d-\epsilon})$ if co-NP $\not\subseteq$ NP/poly.

The following Corollary follows from Theorem 6 and Lemma 4.

Corollary 7. Point Hyperplane Cover *in* $\mathbb{R}^d$ *has a kernel of size* $\Theta(k^d)$ *if* co-NP $\not\subseteq$ NP*/poly.*

Using similar techniques, we also obtain tight kernels for Projective Point Hyperplane Cover in $\mathbb{R}^d$.

Lemma 8. Projective Point Hyperplane Cover *in* $\mathbb{R}^d$ *has a kernel of size* $\Theta(k^{d-1})$ *if* co-NP $\not\subseteq$ NP*/poly.*

By the method suggested by Dell an Melkebeek [8], we can show a lower bound on the number of points of a polynomial kernel for Point Hyperplane Cover in $\mathbb{R}^d$, for each fixed positive integer d.

Lemma 9. Point Hyperplane Cover *in* $\mathbb{R}^d$ *cannot have a kernel with* $\mathcal{O}(k^{d-\epsilon})$ *points if* co-NP $\not\subseteq$ NP*/poly.*

Since Point Hyperplane Cover and Hyperplane Point HS are equivalent problems, we obtain the following corollary.

Corollary 10. Hyperplane Point HS *in* $\mathbb{R}^d$ *cannot have a kernel with* $\mathcal{O}(k^{d-\epsilon})$ *hyperplanes if* co-NP $\not\subseteq$ NP*/poly.*

We also obtain the following corollary.

Corollary 11. Projective Point Hyperplane Cover *in* $\mathbb{R}^d$ *cannot have a kernel with* $\mathcal{O}(k^{d-1-\epsilon})$ *points if* co-NP $\not\subseteq$ NP*/poly.*

4 Covering Polynomials of Bounded Degree with Points

In this section, we consider the D-Polynomial Point HS problem and show that this problem is equivalent to Hyperplane Point HS in a higher dimensional space. We utilize this to give tight polynomial kernels for D-Polynomial Point HS, when the underlying vector space of polynomials is α-good.

Recall that in D-Polynomial Point HS, a vector space $\mathcal{R}$ of D-degree polynomials in $\text{Poly}_D[X_1, X_2, \ldots, X_d]$ is specified. The input is a set $\mathcal{F}$ of n polynomials from $\mathcal{R}$ and the objective is to find at most k points in $\mathbb{R}^d$ that cover all the input polynomials.

We utilize the Veronese mapping from a vector space of D-degree polynomials to the subsystem of hyperplanes in Euclidean space $\mathbb{R}^b$. Such a mapping is a bijective mapping between the vector space of D-degree polynomials and the hyperplanes in $\mathbb{R}^b$. However, the mapping need not be an onto mapping from $\mathbb{R}^d$ to $\mathbb{R}^b$. Let $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$ be the image of $\mathbb{R}^d$ under the Veronese mapping $\Phi_{\mathcal{R}}$. Thus, $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d) \subseteq \mathbb{R}^b$. We show that the Hyperplane Point HS problem for an α-good vector space $\mathcal{R}$ in $\mathbb{R}^b$ when the solution set is restricted to belonging to $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$, does not have a $\mathcal{O}(k^{b-\epsilon})$ kernel unless co-NP $\subseteq$ NP/poly.

Before this, we require a few results regarding the behaviour of points under the Veronese mapping.

First, we show that a set of n points in $\mathbb{R}^d$ that are in general position with respect to $\mathcal{R}$ are mapped to a set of n points in $\mathbb{R}^b$ in general position with respect to hyperplanes in $\mathbb{R}^b$.

Claim 4. *Let $\mathcal{P}$ be a set of points in $\mathbb{R}^d$, and $\mathcal{R}$ be a subspace of* Poly_D *$[X_1, \ldots, X_d]$ with a basis $\{f_1(X), \ldots, f_b(X), 1\}$ where $X = (X_1, \ldots, X_d)$.*

1. *If the set $\mathcal{P}$ is in general position with respect to the polynomial family $\mathcal{R}$ then the image $\Phi_{\mathcal{R}}(\mathcal{P})$, under the Veronese mapping $\Phi_{\mathcal{R}}$, is a $|\mathcal{P}|$-sized set in general position with respect to hyperplanes in $\mathbb{R}^b$.*
2. *Let $S = \{q_1, \ldots, q_\ell\} \subseteq \Phi_{\mathcal{R}}(\mathcal{P})$ be in general position with respect to hyperplanes in $\mathbb{R}^b$. Then the set $S' = \{p_1, \ldots, p_\ell\}$, where $p_i \in \Phi_{\mathcal{R}}^{-1}(q_i) \cap \mathcal{P}$, will be a $|S|$-sized set in general position with respect to $\mathcal{R}$.*

Proof. 1. First, observe that if the map $\Phi_{\mathcal{R}}$ is injective on $\mathcal{P}$ then the result will directly follow. However, in general, the map $\Phi_{\mathcal{R}}$ need not be an injective mapping on an arbitrary set of n points in $\mathbb{R}^d$. We show that $\Phi_{\mathcal{R}}$ is injective when restricted to $\mathcal{P}$ if $\mathcal{P}$ is in general position with respect to $\mathcal{R}$. To reach a contradiction, let $\Phi_{\mathcal{R}}(p_1) = \Phi_{\mathcal{R}}(p_2)$ where $p_1, p_2\,(\neq p_1) \in \mathcal{P}$. Let $S \subseteq \mathcal{P}$ be of size $b+1$ and $p_1, p_2 \in \mathcal{P}$. Observe that the set $\Phi_{\mathcal{R}}(S)$ will have less than $b+1$ points and this will imply that there exists a hyperplane $\sum_{i=1}^{b} \lambda_i Z_i + \lambda_{b+1} = 0$ on which the set $\Phi_{\mathcal{R}}(S)$ will lie. But this implies that the polynomial $\sum_{i=1}^{b} \lambda_i f_i(X) + \lambda_{b+1} = 0$ will be satisfied by all the points in S. Thus, we have reached a contradiction from the fact that the point set $\mathcal{P}$ was in general position.
2. The second part of the Claim follows directly from the construction of the mapping $\Phi_{\mathcal{R}}$.

Next, for each n we construct a set of n points that satisfy the conditions of Lemma 5 and belong to $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$, where $\mathcal{R}$ is α-good. This construction is mainly done in $\mathbb{R}^d$ and follows exactly along the lines of the proof of Lemma 5.

Lemma 12. *Let $\mathcal{R}$ be a α-good vector space of D-degree polynomials in $\mathbb{R}^d$ and let the Veronese mapping $\Phi_{\mathcal{R}}$ linearize $\mathcal{R}$ into $\mathbb{R}^b$. Let $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$ be the image of $\Phi_{\mathcal{R}}$. Then for every $n \in \mathbb{Z}^+$, there is a* poly(n) *time algorithm to construct a set $\mathcal{P}$ of n points in $\mathbb{R}^b$ that have the following properties:*

(1) The points are in general position with respect to hyperplanes in $\mathbb{R}^b$.
(2) Let $\mathcal{H}$ be the family of hyperplanes defined by each set of b points from $\mathcal{P}$. The hyperplanes in the family $\mathcal{H}$ are in general position, i.e., given r hyperplanes $H_1, \ldots, H_r$ in $\mathcal{H}$ with $r \leq b$ the dimension of the affine space $\cap_{i=1}^r H_r$ is $b - r$.
(3) For any point p in $\mathbb{R}^b \backslash \mathcal{P}$, there are at most b hyperplanes in $\mathcal{H}$ that contain p.

Proof. As in the proof of Lemma 5, we will construct the set $\mathcal{P}$ inductively. We start with a set $\mathcal{P}_b$ of size b such that the set is in general position with respect to $\mathcal{R}$. This can be constructed in $O(1)$ time as $\mathcal{R}$ is α-good. We then extend this set one point at a time using points from the grid (as in the proof of Lemma 5). Assume that for $b \leq t < n$, we have constructed a point set $\mathcal{P}_t$ that satisfies the above conditions. The points forbidden to be added to the set $\mathcal{P}_t$ will lie on a bounded number of polynomials from $\mathcal{R}$ and we will call these polynomials *forbidden polynomials.* The hyperplane that is in bijective correspondence with a forbidden polynomial under the Veronese mapping $\Phi_{\mathcal{R}}$ is called a *forbidden hyperplane.* As in the proof of Lemma 4, we can show, using the Veronese mapping $\Phi_{\mathcal{R}}$, that the number of forbidden hyperplanes arising due to conditions (1), (2) and (3) is bounded by $O(t^b)$, $O(t^{b^2+b-1})$ and $O(t^{b^2+b-1})$ respectively. This also gives a bound on the number of forbidden polynomials.

As we have an upper bound on the number of forbidden polynomials, we can now use the same trick to generate points satisfying conditions (1) to (3) as Lemma 4. In this case we take a d-dimensional $m \times \cdots \times m$ grid with $m = n^{b^2+b}$ and use the fact that given any polynomial from $\mathcal{R}$, the number of points of the grid hitting it is bounded by $m^{d-\alpha}$. This completes the proof. $\square$

This helps us to prove a kernel lower bound on the restricted version of Hyperplane Point Cover described above.

Lemma 13. *Let $\mathcal{R}$ be an α-good vector space of D-degree polynomials in $\mathbb{R}^d$ and let the Veronese mapping $\Phi_{\mathcal{R}}$ linearize $\mathcal{R}$ into $\mathbb{R}^b$. Then* Hyperplane Point Cover*, when the solution is restricted to belong to $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d) = \Phi_{\mathcal{R}}(\mathbb{R}^d)$, cannot have a kernel of size $\mathcal{O}(k^{b-\epsilon})$ unless* co-NP $\subseteq$ NP*/poly.*

Proof. The construction of n points in $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$ described in Lemma 12 has all the properties described in Lemma 5. The rest of the proof follows exactly as the proof of Theorem 6.

Finally, the following Theorem is derived from Lemma 13 by utilizing the Veronese mapping $\Phi_{\mathcal{R}}$.

Theorem 14. *D-Polynomial Point HS for an α-good vector space $\mathcal{R}$ in $\mathbb{R}^d$, and having the Veronese mapping into $\mathbb{R}^b$, (i) has a polynomial kernel of size $\mathcal{O}(k^b)$, (ii) does not have a polynomial kernel of size $\mathcal{O}(k^{b-\epsilon})$, unless* co-NP $\subseteq$ NP*/poly.*

Proof. To prove the tightness of a $\mathcal{O}(k^b)$ kernel for D-Polynomial Point HS in $\mathbb{R}^d$ with the Veronese mapping into $\mathbb{R}^b$, first we prove (i) by giving an upper bound on the size of a kernel. Let the polynomials in an instance of D-Polynomial Point HS come from the vector space $\mathcal{R}$, as defined earlier. The Veronese mapping $\Phi_{\mathcal{R}}$ is a reduction from D-Polynomial Point HS in $\mathbb{R}^d$ to Hyperplane Point HS in $\mathbb{R}^b$. Thus, since Hyperplane Point HS in $\mathbb{R}^b$ has a $\mathcal{O}(k^b)$ kernel [17], so does D-Polynomial Point HS in $\mathbb{R}^d$ with the Veronese mapping into $\mathbb{R}^b$.

To prove (ii), we use the Veronese mapping on the vector space $\mathcal{R}$ of D-degree polynomials more carefully. Let the hyperplanes, to which the polynomials are mapped, be in $\mathbb{R}^b$. The mapping is a bijective function. Thus, in order to obtain the required result, we give a reduction from Hyperplane Point HS in $\mathbb{R}^b$, where the solution set of points come from $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$. The reduction is simply the reverse function of the Veronese mapping. Suppose an instance $(\mathcal{H}, k)$ of Hyperplane Point HS where the solution set of points belong to $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$ reduces to the instance $(\mathcal{H}', k)$ of D-Polynomial Point HS. If $(\mathcal{H}, k)$ is a Yes instance, then there is a set S of at most k points in $\mathbb{R}^b$ that covers all the hyperplanes in $\mathcal{H}$. Consider the set S' of points in $\mathbb{R}^d$ by taking one preimage of each point in S. The set S' is exactly the same size as S, and therefore contains at most k points. Moreover, by definition of the Veronese mapping, S' covers all the polynomials in $\mathcal{H}'$. Therefore, $(\mathcal{H}', k)$ is also a Yes instance for D-Polynomial Point HS.

On the other hand, if $(\mathcal{H}', k)$ is a Yes instance of D-Polynomial Point HS, then there is a set S' of at most k points that cover all the polynomials in $\mathcal{H}'$. The image of S' under the Veronese mapping will be of size at most S' and will cover the family $\mathcal{H}$. Therefore, $(\mathcal{H}, k)$ will be a Yes instance of Hyperplane Point Cover when the solution points can come only from $\mathsf{Ver}_{\mathcal{R}}(\mathbb{R}^d)$. Thus, by Lemma 13, we conclude that D-Polynomial Point HS cannot have a kernel of size $\mathcal{O}(k^{b-\epsilon})$ unless co-NP $\subseteq$ NP/poly.

5 Open Problems

The D-Point Polynomial Cover problem in $\mathbb{R}^d$ requires a set of n points in $\mathbb{R}^d$ to be covered by at most k D-degree polynomials. Although polynomial kernels for D-Point Polynomial Cover in $\mathbb{R}^d$ can be exhibited, tight lower bounds for this problem are unknown.

References

1. Afshani, P., Berglin, E., van Duijn, I., Nielsen, J.S.: Applications of incidence bounds in point covering problems. In: SoCG, pp. 60:1–60:15 (2016)
2. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. TCS **237**(1–2), 123–134 (2000)
3. Ashok, P., Kolay, S., Misra, N., Saurabh, S.: Unique covering problems with geometric sets. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 548–558. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_43
4. Ashok, P., Kolay, S., Saurabh, S.: Multivariate complexity analysis of geometric red blue set cover. Algorithmica **79**, 1–31 (2015)
5. Boissonnat, J., Dutta, K., Ghosh, A., Kolay, S.: Kernelization of the subset general position problem in geometry. In: MFCS, pp. 25:1–25:13 (2017)
6. Bringmann, K., Kozma, L., Moran, S., Narayanaswamy, N.S.: Hitting set for hypergraphs of low VC-dimension. In: ESA, pp. 23:1–23:18 (2016)
7. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms, vol. 3. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
8. Dell, H., Van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: STOC, pp. 251–260. ACM (2010)
9. Dinur, I., Steurer, D.: Analytical approach to parallel repetition. In: STOC, pages 624–633. ACM (2014)
10. Dom, M., Fellows, M.R., Rosamond, F.A.: Parameterized complexity of stabbing rectangles and squares in the plane. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 298–309. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00202-1_26
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0515-9. 530 p.
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, New York Inc., Secaucus (2006). https://doi.org/10.1007/3-540-29953-X
13. Giannopoulos, P., Knauer, C., Whitesides, S.: Parameterized complexity of geometric problems. Comput. J. **51**(3), 372–384 (2008)
14. Heggernes, P., Kratsch, D., Lokshtanov, D., Raman, V., Saurabh, S.: Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. Inf. Comput. **231**, 109–116 (2013)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Jünger, M., et al. (eds.) 50 Years of Integer Programming 1958–2008 - From the Early Years to the State-of-the-Art, pp. 219–241. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-540-68279-0_8
16. Kratsch, S., Philip, G., Ray, S.: Point line cover: the easy kernel is essentially tight. TALG **12**(3), 40 (2016)
17. Langerman, S., Morin, P.: Covering things with things. DCG **33**(4), 717–729 (2005)
18. Matoušek, J.: Lectures on Discrete Geometry, vol. 212. Springer Science & Business Media, New York (2002). https://doi.org/10.1007/978-1-4613-0039-7
19. Megiddo, N., Tamir, A.: On the complexity of locating linear facilities in the plane. Oper. Res. Lett. **1**(5), 194–197 (1982)
20. Vazirani, V.V.: Approximation Algorithms. Springer Science & Business Media, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04565-7

A Tight Bound for Shortest Augmenting Paths on Trees

Bartłomiej Bosek[1], Dariusz Leniowski[2], Piotr Sankowski[2], and Anna Zych-Pawlewicz[2](✉)

[1] Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
bosek@tcs.uj.edu.pl

[2] Institute of Computer Science, University of Warsaw, Warsaw, Poland
{d.leniowski,sank,anka}@mimuw.edu.pl

Abstract. The shortest augmenting path technique is one of the fundamental ideas used in maximum matching and maximum flow algorithms. Since being introduced by Edmonds and Karp in 1972, it has been widely applied in many different settings. Surprisingly, despite this extensive usage, it is still not well understood even in the simplest case: online bipartite matching problem on trees. In this problem a bipartite tree $T = (W \uplus B, E)$ is being revealed online, i.e., in each round one vertex from B with its incident edges arrives. It was conjectured by Chaudhuri et al. [7] that the total length of all shortest augmenting paths found is $O(n \log n)$. In this paper we prove a tight $O(n \log n)$ upper bound for the total length of shortest augmenting paths for trees improving over $O(n \log^2 n)$ bound [5].

1 Introduction

One of the most fundamental techniques used to solve maximum matchings or flow problems is the augmenting path technique. It augments the solution along residual paths until the maximum size matching/flow is found. Intuitively, the work needed for that should be minimized if shortest paths are chosen each time. In particular, this was the key concept that allowed Edmonds and Karp in 1972 to show the first strongly polynomial time algorithm for the maximum flow problem [9]. Since then it has been widely applied. Surprisingly, despite this effort, it is still not well understood even in the simplest case—online bipartite matching problem on trees. This may be due to the fact that shortest augmenting paths do not seem to have strong enough structure admitting exact analysis. Other methods for choosing augmenting paths are easier to analyze [4,7]. Our

The work of all authors was supported by Polish National Science Center grant 2013/11/D/ST6/03100. Additionally, the work of P. Sankowski was partially supported by the project TOTAL (No. 677651) that has received funding from ERC.

work is meant as a step forward towards understanding the shortest augmenting path method for computing the matching on bipartite graphs.

To be able to analyze this approach we adopt the following model. Let W and B be the bipartition of vertices over which the tree will be formed. The set W (called white vertices) is given up front to the algorithm, whereas the vertices in B (black vertices) arrive online. We denote by $F_t = \langle W \uplus B_t, E_t \rangle$ the forest after the t'th black vertex has arrived where $X \uplus Y$ is a disjoint sum of X and Y. The graphs F_t for $t \in [n] = \{1, \ldots, n\}$ are constructed online in the following manner. We start with $F_0 = \langle W \uplus B_0, E_0 \rangle = \langle W \uplus \emptyset, \emptyset \rangle$. In turn $t \in [n]$ a new vertex $b_t \in B$ together with all its incident edges $E(b_t)$ is revealed and F_t is defined as: $E_t = E_{t-1} \cup E(b_t)$ and $B_t = B_{t-1} \cup \{b_t\}$. In the model we consider, none of the newly added edges is allowed to close the cycle. For simplicity we assume that we add in total $n = |W|$ black vertices. The final graph is a tree denoted as $F_n = (W \uplus B_n, E_n)$.

The goal of the online algorithm is to compute for each F_t the maximum size matching M_t, possibly making use of M_{t-1}. In this paper we study one specific algorithm, referred to as the Shortest Augmenting Path algorithm. When b_t arrives, the Shortest Augmenting Path algorithm always chooses the shortest among all available augmenting paths. A natural question that we ask is what is the total length of all paths applied by the Shortest Augmenting Path algorithm. In this paper the unmatched vertices are referred to as *free*. For a vertex v we denote its neighborhood in F_t as $\mathrm{N}_t(v)$. By $F[X] = \langle X, E(X) \rangle$ we denote a subgraph of F induced by $X \subseteq W \cup B$, where $E(X) = \{e \in E : e \subseteq X\}$.

2 Motivation and Related Work

The online bipartite matching problem with augmentations has recently received increasing attention [1,4,5,7,10,11]. The model we study has been introduced in [10]. As mentioned before, the key point of this model is to focus on bounding the total length of augmenting paths and not the running time of the algorithm. This is motivated as follows. Imagine that the white vertices are servers and black vertices are clients. The clients arrive online. A typical client may be a portable computing box, trying to connect to a huge network of services with some specific request. The edges of the graph reflect eligibility of the servers to answer clients request. The classical online model (as in [3,8,13]) does not allow preemption, i.e., the client cannot change the server. In such setting one must accept some clients not being served while they could possibly be served with preemption. In that model a famous ranking algorithm gives an optimal $(1-1/e)$-approximation [13]. The authors in [10] wonder if preemption makes sense. It may be beneficial to reallocate clients provided that only a limited number of reallocations is needed. This leads to the question of how many reallocations are needed if one insists on serving every client. In [10] a special case is studied when each client can connect to at most two servers. In such scenario the authors prove that the Shortest Augmenting Path algorithm performs $\mathcal{O}(n \log n)$ reallocations and that no algorithm can do better than that. Chaudhuri et al. [7] show that the

Shortest Augmenting Path algorithm makes a total of $\mathcal{O}(n \log n)$ reallocations in the case of general bipartite graph, provided that the clients arrive in a random order. They conjecture, however, that this should be the case also for the worst case arriving order of clients. Until this paper, this conjecture remained open even for trees. In [5] the authors prove a bound of $\mathcal{O}(n \log^2 n)$ for Shortest Augmenting Path algorithm given that the underlying graph is a tree. In this paper we take a different approach and prove the conjecture of Chaudhuri et al. for trees. In this restricted case, the authors of [7] proposed a different augmenting path algorithm that achieves total paths' length of $\mathcal{O}(n \log n)$. Their algorithm, however, is only applicable to trees. The Shortest Augmenting Path algorithm, on the other hand, applies to any bipartite graph and also is very simple. This is the reason why we feel it is important to study this algorithm. Our ultimate goal is to show the bound of $\mathcal{O}(n \log n)$ for general bipartite graphs. We believe that the techniques proposed in this paper are an important step forward on the path to achieve this goal. In parallel work to ours [1] the authors provide a bound of $\mathcal{O}(n \log^2 n)$ total number of reallocations for the Shortest Augmenting Path algorithm on general bipartite graphs. This recent result has been accepted to *SODA 2018* and it nearly closes the conjecture of Chaudhuri et al. We note, however, that their techniques alone do not lead to $\mathcal{O}(n \log n)$ even for trees. Before this result, for general graphs, nothing interesting was known for Shortest Augmenting Path algorithm. A different algorithm was proposed achieving much worse $\mathcal{O}(n\sqrt{n})$ bound on the total length of augmenting paths [4].

Our model is strongly related to dynamic algorithms. There, we are not only interested in constructing short augmenting paths. An efficient way of finding them is the most important aspect. Most papers in this area consider edge updates in a general fully-dynamic model which allows for insertions and deletions of edges intermixed with each other. This is a much more difficult scenario in which one cannot do much when constrained by our model. In particular, if edges are added to a bipartite graph, one can show an instance for which any algorithm maintaining a maximum matching performs $\Omega(n^2)$ reallocations. Hence, it is reasonable to stop insisting on matching every client and accept approximate solutions. Here we want to approximate the maximum matching size and not the number of reallocations. One also needs to keep in mind that a trivial greedy algorithm maintaining a maximal matching gives a 1/2-approximation and preforms no reallocations at all. A 2/3-approximation algorithm by [14] achieves $\mathcal{O}(\sqrt{m})$ update time. Gupta and Peng give a $(1-\varepsilon)$-approximation in $\mathcal{O}(\sqrt{m}\varepsilon^{-2})$ time per update [12]. The $\mathcal{O}(\sqrt{m})$ barrier was broken by Bernstein and Stein who gave a $(\frac{2}{3}-\varepsilon)$-approximation algorithm that achieves $O(m^{1/4}\varepsilon^{-2.5})$ update time [2]. Finally, $(1-\varepsilon)$ approximation in $O(m\varepsilon^{-1})$ total time and with $O(n\varepsilon^{-1})$ total length of paths was shown in [4] in a model most related to ours, i.e., when vertices are added on one side of the bipartition. There are also randomized algorithms in the dynamic model [15] maintaining the exact size of a maximum matching with $\mathcal{O}(n^{1.495})$ update time. They do not imply any bound on the number of changes to the matching as they use algebraic techniques that are not based on augmenting paths.

3 The Mini-Max Game

Our goal in this paper is to prove that the total length of all augmenting paths applied by Shortest Augmenting Path algorithm on a tree is $O(n \log n)$. More formally, we want to prove the following, where by $\|\pi\|$ we denote the number of edges on a path π.

Theorem 1. *Let π_t be the path applied by Shortest Augmenting Path algorithm in turn t. Then $\sum_{t=1}^{n} \|\pi_t\| \in O(n \log n)$.*

The idea is not to study directly the paths applied by Shortest Augmenting Path algorithm, but a collection of other paths that are possibly longer. To be more precise, we model a scenario where in each turn Shortest Augmenting Path algorithm gets the worst possible matching on F_t (i.e., the one maximizing the shortest augmenting path). We then study the worst case augmenting paths rather then the ones given by the matching produced by Shortest Augmenting Path algorithm. Interestingly, these paths can be defined without mentioning any matching. In this section we provide the appropriate definitions and show that they work as expected.

Let us consider what worst possible matching could there be. Think of a game, where the algorithm chooses a shortest augmenting path, and the adversary chooses a matching where such path is the longest. We are given graph F_t and the newly presented vertex b_t. We are interested in a matching where b_t is not matched, so that we model the worst case matching before b_t is matched. The game starts in vertex b_t, where the algorithm may choose which edge to follow among the unmatched edges incident to b_t. Then the algorithm stumbles upon a white vertex where it has to follow the matching edge chosen by the adversary. The game continues until a leaf is reached (either black or white, black meaning that the algorithm did not find a path). It is not hard to see that the algorithm, when it has a choice, wants to minimize the distance to a free white vertex, while the adversary tries to maximize it. This way we obtain a two-person game, where the outcome of the game is the length of the shortest augmenting path. If the path does not exist, we let the outcome be infinite. Throughout the paper we let $\infty + 1 = \infty$ and we write $x < \infty$ to indicate that x is simply an integer.

We move on to stating formal definitions. We start by introducing our game on any rooted tree T whose vertices are either black or white. We then define the outcome of the algorithm player in time t for a specific T closely related to F_t. For a rooted tree T we denote the list of children of a vertex v in T as $\mathrm{Ch}_T(v)$ and a parent of v as $\mathrm{parent}_T(v)$.

Definition 1. *Let T be a rooted tree whose vertices are partitioned into two sets: $V(T) \subseteq B \uplus W$. For each $b \in B$ we define its revenue as* $\text{mini-max}_T(b) = \min_{w \in \mathrm{Ch}_T(b)} \text{mini-max}_T(w) + 1$ *if* $\mathrm{Ch}_T(b) \neq \emptyset$ *and* $\text{mini-max}_T(b) = \infty$ *otherwise. For each $w \in W$ we define its revenue as* $\text{mini-max}_T(w) = \max_{b \in \mathrm{Ch}_T(w)} \text{mini-max}_T(b) + 1$ *if* $\mathrm{Ch}_T(w) \neq \emptyset$ *and* $\text{mini-max}_T(w) = 0$ *otherwise. We let* $\text{mini-max-next}_T(v)$ *be the child of v whose revenue determines the minimum or*

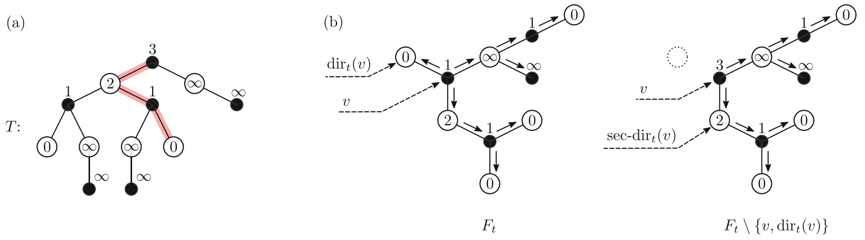


Fig. 1. (a) Example of a rooted mini-max tree with vertex revenues. (b) Example of mini-max distances for a vertex v in turn t.

the maximum respectively.[1] *If* v *has no children,* $\text{mini-max-next}_T(v)$ *is undefined. We define the mini-max path starting in a vertex* v *as* $\text{mini-max-path}_T(v) = v \cdot \text{mini-max-path}_T(\text{mini-max-next}_T(v))$ *if* $\text{mini-max-next}_T(v)$ *is defined and* $\text{mini-max-path}_T(v) = v$ *otherwise.*[2]

Definition 1 is illustrated by example in Fig. 1(a). Based on this definition we can define the first and the second mini-max distance from a given vertex to a white leaf in a specific time moment t. In addition to that we define the first and second direction, i.e., the vertex one needs to follow to find the first and second mini-max distance.

Definition 2. *Let* $t \in [n]$ *and* $v \in B_t \cup W$*. Let* T *be the connected component of* v *in* F_t *rooted in* v*. Let* $\text{dist}_t(v) = \text{mini-max}_T(v)$ *and* $\text{dir}_t(v) = \text{mini-max-next}_T(v)$*. Let now* S *be a rooted tree, where from* T *we remove* $\text{mini-max-next}_T(v)$ *and all its descendants. Let* $\text{sec-dist}_t(v) = \text{mini-max}_S(v)$ *and* $\text{sec-dir}_t(v) = \text{mini-max-next}_S(v)$*.*

Definition 2 is illustrated by example in Fig. 1(b). We next observe the monotonicity of the mini-max distance functions we defined.

Observation 2 (Appendix A in [6])**.** *Fix a vertex* $u \in B \cup W$*. The functions* $\text{dist}_t(u)$ *and* $\text{sec-dist}_t(u)$ *are non-decreasing with respect to* t *for the whole range of* t *where* $u \in V(F_t)$*.*

We intuitively explained how the mini-max distances correspond to the augmenting paths of the worst case matching, so the hope is that they bound from above the augmenting paths applied by Shortest Augmenting Path algorithm. The next lemma shows that this intuition is reflected in reality. It states that no matter what matching is given on F_t for some $t \in [n]$, the path chosen by Shortest Augmenting Path algorithm to match b_t is bounded by $\text{dist}_t(b_t)$.

[1] If there are more such vertices we choose the first one according to some predefined order on $B \cup W$.

[2] Symbol $\cdot$ denotes concatenation of paths.

Lemma 3 (Appendix A in [6]). *Let $1 \leqslant t_0 \leqslant t \leqslant n$ and let ρ_t be the shortest augmenting path from b_{t_0} to a free white vertex according to any given matching M in F_t where b_{t_0} is free. It holds that if $\mathrm{dist}_t(b_{t_0}) < \infty$ then ρ_t exists and $\|\rho_t\| \leqslant \mathrm{dist}_t(b_{t_0})$.*

4 Dead Vertices

In this section we introduce another concept crucial for our result. We define here *dead* vertices and give some intuition why this makes sense. In fact dead vertices reflect the infinity of some mini-max distance functions. For completeness, in addition to defining dead vertices, we describe the situations when the mini-max distance functions are infinite. We start with the statements.

Definition 3. *A vertex $b_{t_0} \in B_t$ breaks Hall's condition in time $t \geqslant t_0$ iff $b_{t_0} \in X$ for some inclusion-wise minimal set X satisfying $|\mathrm{N}_t(X)| < |X|$.*

Lemma 4 (Appendix B in [6]). *Let $1 \leqslant t_0 \leqslant t \leqslant n$. Then $\mathrm{dist}_t(b_{t_0}) = \infty$ iff b_{t_0} breaks Hall's condition in time t.*

Corollary 5. *If $\mathrm{dist}_t(b_t)$ is infinite, and we are given some maximum matching M_{t-1} for F_{t-1}, then there is no augmenting (with respect to M_{t-1}) path for b_t in time t.*

We now move on to defining dead vertices. The definitions may not seem very intuitive, but we provide some intuition shortly after introducing them.

Definition 4. *We say that a vertex $b_{t_0} \in B$ is* dead *in turn $t \geqslant t_0$ iff $\text{sec-dist}_t(b_{t_0}) = \infty$.*

Definition 4 combined with Lemma 4 implies, that any black vertex that breaks Hall's condition in time t is dead in time t, but not necessarily the other way around.

Definition 5. *A white vertex $w \in W$ is* dead *in time t iff $\mathrm{dist}_t(w) = \infty$.*

We say that a vertex is alive iff it is not dead. We denote as A_t the set of vertices of $B_t \cup W$ that are alive in turn t and as D_t the set of vertices of $B_t \cup W$ that are dead in turn t. If $v \in A_{t-1} \cap D_t$, we say that v dies in turn t. Note that due to monotonicity (Observation 2), once a vertex dies, it never comes back alive. The following observations bring some intuition into the picture of dead versus alive vertices. Observation 6 given below follows from Definitions 2, 4 and 5.

Observation 6. *1. A black leaf is dead from the moment it arrives.*
2. A black vertex is dead iff it has at most one alive neighbour.
3. A white vertex is dead iff it has at least one dead neighbour.

The intuition behind dead vertices is that they determine regions of F_t where Hall's condition is either broken or tight. The mini-max paths in turn t that correspond to finite mini-max distances do not visit vertices that were dead in turn $t-1$. Moreover, if a mini-max path in F_t (whose corresponding mini-max distance is finite) enters a vertex that is alive in turn t, it does not visit anymore vertices dead in turn t. We state this formally as Lemma 7. This reflects the behavior of augmenting paths. If a maximum matching is maintained, then the augmenting path from turn t does not enter the regions where Hall's condition is tight in F_{t-1}.

Lemma 7 (Appendix B in [6]). *Let $t \in [n]$ and $v \in A_t$. Pick any vertex as a root of the connected component of v in F_t and let T be the corresponding rooted tree. Then $V(\text{mini-max-path}_T(v)) \subseteq A_t$.*

In the remainder of this section we specify precisely which vertices die in turn t. The first lemma does not describe who dies or stays alive, but it is an important complement of the subsequent two lemmas, which cover all the situations when vertices die.

Lemma 8 (Appendix B in [6]). *If b_t does not break Hall's condition in turn t, then b_t has at least one neighbour in F_t which was alive in turn $t-1$.*

So if a black vertex added in turn t does not break Hall's condition, then it has at least one neighbour who was alive in turn $t-1$. The next two lemmas cover two cases. The first lemma states that if the new black vertex has at least two such neighbours, then no vertices die in turn t. If, however, it has exactly one such neigbour, then some vertices die in turn t and the second lemma describes precisely which ones.

Lemma 9 (Appendix B in [6]). *If b_t has at least two neighbours which are alive in turn $t-1$ then b_t is alive in turn t and no vertex dies in turn t.*

The last lemma covers the only case when vertices die in turn t. It shows that there is a certain region around b_t where vertices die, and a barrier for that region are special vertices called life portals, defined below. The picture illustrating which region dies in turn t is given in Fig. 2.

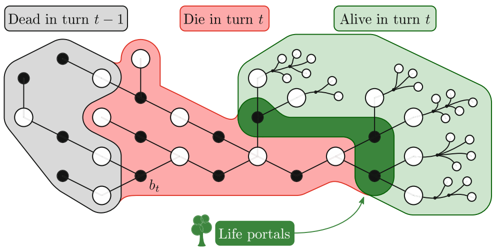


Fig. 2. The vertices that die in turn t.

Definition 6. *A black vertex b is a life portal in turn t iff $|\mathrm{N}_t(b) \cap A_{t-1}| \geqslant 3$. The set of life portals in turn t is denoted as $\mathcal{LP}_t$.*

Lemma 10 (Appendix B in [6]). *If b_t has exactly one neighbour in F_t which was alive in turn $t-1$ and there is a path π from b_t to $v \in B_t \cup W$ such that (1) all vertices of π were alive in turn $t-1$ and (2) there are no life portals from $\mathcal{LP}_t$ on π, then v dies in turn t. Vertices of $B_t \cup W$ that cannot be reached from b_t via such path do not die in turn t.*

Corollary 11. *Lemma 10 shows, that for all t such that b_t does not break Hall's condition and has exactly one neighbour alive in turn $t-1$, statement $|\mathrm{N}_t(b) \cap A_{t-1}| \geqslant 3$ in Definition 6 is equivalent to $|\mathrm{N}_t(b) \cap A_t| \geqslant 2$.*

5 The Proof

In the remainder of the paper we present the proof of Theorem 1, which states that if π_t is the path applied by Shortest Augmenting Path algorithm in turn t, then $\sum_{t=1}^{n} \|\pi_t\| \in \mathcal{O}(n \log n)$. As we mentioned in Sect. 3, we do not study $\|\pi_t\|$ directly. Instead, we want to study distance functions $\mathrm{dist}_t(b_t)$ introduced in Sect. 3. By Lemma 3 given in Sect. 3 we know that $\mathrm{dist}_t(b_t)$ bounds $\|\pi_t\|$ from above. Recall that $\mathrm{dist}_t(b_t)$ is the mini-max distance from b_t to a white leaf in F_t. By definition if $\mathrm{dist}_t(b_t) < \infty$, then there is a path from b_t to a white leaf which certifies it. We introduce the formal definition of such path below.

Definition 7. *Let $t \in [n]$ and $v \in B_t \cup W$. Let T be a connected component of v in F_t rooted at v. Then $\mathrm{path}_t(v) = \text{mini-max-path}_T(v)$.*

Note that by Definitions 1 and 2, if $\mathrm{dist}_t(b_t) < \infty$, then $\|\mathrm{path}_t(b_t)\| = \mathrm{dist}_t(b_t)$. In addition to that, we define a path that certifies that $\text{sec-dist}_t(b_t)$ is finite.

Definition 8. *Let $t \in [n]$ and $v \in B_t \cup W$. Let S be a connected component of v in $F_t - \{v, \mathrm{dir}_t(v)\}$ rooted at v, where $F_t - \{v, \mathrm{dir}_t(v)\}$ denotes F_t with edge $\{v, \mathrm{dir}_t(v)\}$ removed. Then $\text{sec-path}_t(v) = \text{mini-max-path}_S(v)$.*

Again by Definitions 1 and 2, if $\text{sec-dist}_t(b_t) < \infty$, then $\|\text{sec-path}_t(b_t)\| = \text{sec-dist}_t(b_t)$. Instead of proving Theorem 1, in the remainder of this paper we prove that $\sum_{t:\mathrm{dist}_t(b_t)<\infty} \|\mathrm{path}_t(b_t)\| \in \mathcal{O}(n \log n)$. This is in fact a stronger statement. We claim that even if the adversary picks the worst possible maximum matching in each turn, the Shortest Augmenting Path algorithm still applies paths of total length $\mathcal{O}(n \log n)$. Note that if $\mathrm{dist}_t(b_t) = \infty$, then due to Corollary 5 Shortest Augmenting Path algorithm cannot match the new vertex b_t if the maximum matching is given on the remaining vertices. Our proof of such simple statement is unfortunately rather complex. Before we move on to it, we give some intuitions on where the actual problem hides. It is enlightening to discover, that with the additional assumption that the black vertices are of degree two or more, the statement above has a very simple proof.

Lemma 12. *If each black vertex b_t has degree at least 2, then* $\sum_{t=1}^{n} \|\mathrm{path}_t(b_t)\| \leqslant n \log_2 n$.

Proof. We start by observing that no vertex ever dies. In turn $t = 0$ the only presented vertices are W, so by definition all vertices are alive in turn $t = 0$. Let $t > 0$ and assume that no vertices died up until turn $t - 1$. Due to Lemma 9 no vertex dies in turn t and b_t is alive in turn t. This implies that $\mathrm{dist}_t(b_t) < \infty$ and $\text{sec-dist}_t(b_t) < \infty$. Hence, $\mathrm{path}_t(b_t)$ and $\text{sec-path}_t(b_t)$ are two separate paths, contained in two different components of F_{t-1} connected in turn t by b_t. Also, $\|\mathrm{path}_t(b_t)\| \leqslant \|\text{sec-path}_t(b_t)\|$. Thus, $\mathrm{path}_t(b_t)$ is at most as long as the size of the smaller of the two components. We pay for $\mathrm{path}_t(b_t)$ by charging 1 token to each vertex in every component but the largest one among the components of F_{t-1} connected by b_t in turn t. A vertex v is charged when v's component size increases at least twice, so v cannot be charged more than $\log_2 n$ times. This gives a total charge of at most $n \log_2 n$. □

The essence of this proof is that every time a black vertex is added, it connects at least two trees into one. As a consequence there are at least two alternative mini-max paths starting from the added vertex, each in a separate tree. The length of the shorter of the two can be charged to the vertices of the smaller tree. If we allow black vertices of degree 1, the situation becomes more complicated, because: (1) there is no alternative path, i.e., the path needs to follow the only edge adjacent to the newly added black vertex, and (2) no trees are merged. Nevertheless the proof of Theorem 1 is a generalization of the proof of Lemma 12. The majority of the remainder of this paper is devoted to addressing issue (2). We define trees which are merged in each turn and allow introducing the charging scheme that generalizes the scheme of Lemma 12. We start, though, by addressing issue (1). To that end we introduce a concept of a dispatching vertex. Even though b_t does not necessarily fork into two alternative mini-max paths, there is another vertex which does. It is the first life portal on $\mathrm{path}_t(b_t)$. We refer to it as dispatching vertex in turn t. To be more formal, we introduce the following definition.

Definition 9. *The dispatching vertex at time $t \in [n]$ is the first black vertex on* $\mathrm{path}_t(b_t)$ *such that* $|\mathrm{N}_t(b) \cap A_t| \geqslant 2$. *We denote it as* $\check{b}_t$.

First observe that $\check{b}_t$ has two alive neighbours in turn t, so there are two alternative mini-max paths branching from $\check{b}_t$. Our next observation is that if $\check{b}_t \neq b_t$, then $\check{b}_t \in \mathcal{LP}_t$: if b_t has two neighbours alive in $t-1$, then due to Lemma 9 no vertex dies in turn t so b_t has two neigbours alive in turn t and hence $b_t = \check{b}_t$; otherwise, if b_t has one neigbour alive in turn $t - 1$, then due to Lemma 10 and Corollary 11 it holds that $\check{b}_t$ is the first life portal of $\mathcal{LP}_t$ on $\mathrm{path}_t(b_t)$. Then also $\check{b}_t$ is the first vertex on $\mathrm{path}_t(b_t)$ that remains alive. All vertices that follow $\check{b}_t$ on $\mathrm{path}_t(b_t)$ remain alive as well. It may happen that $\mathrm{path}_t(b_t)$ contains no life portals, in which case there is no dispatching vertex defined in turn t. This case however is not of concern, since the whole $\mathrm{path}_t(b_t)$ dies in turn t due to Lemma 10. In general, we do not have to worry about vertices that die, and we state this observation as Observation 13 preceded by Definition 10.

Definition 10. *Let $t \in [n]$ be such that $\text{dist}_t(b_t) < \infty$. We let $\text{path}_t(b_t) = \text{path}_t^p(b_t) \cdot \text{path}_t^s(b_t)$, where $\text{path}_t^p(b_t)$ is the prefix of $\text{path}_t(b_t)$ that dies (possibly empty) and $\text{path}_t^s(b_t)$ is the corresponding suffix (also possibly empty).*

Note that if $\check{b}_t$ is defined then $\text{path}_t^s(b_t)$ begins with $\check{b}_t$. Since the final forest has $2n$ vertices and each can die only once, we have the following:

Observation 13. $\sum_{t:\text{dist}_t(b_t)<\infty} \|\text{path}_t^p(b_t)\| \leqslant 2n$.

Thus, to bound $\sum_{t:\text{dist}_t(b_t)<\infty} \|\text{path}_t(b_t)\|$ it suffices to bound $\sum_{t:\text{dist}_t(b_t)<\infty} \|\text{path}_t^s(b_t)\|$. We conclude the list of properties of the dispatching vertex with the following observation.

Observation 14 (Appendix C in [6]**).** *Let $t \in [n]$ and $\text{dist}_t(b_t) < \infty$ and $\check{b}_t$ is defined. Then $\|\text{path}_t^s(b_t)\| = \text{dist}_t(\check{b}_t)$.*

To proceed further, we introduce the crucial notion in our proof: the notion of a level. The levels are some numbers assigned to vertices: each vertex is assigned its level. The intuitive meaning of the level of a vertex is the following. Consider $\text{path}_t(b_t)$, which is the worst case shortest augmenting path for a black vertex b_t added in turn t. For a vertex v, if $\text{path}_t(b_t)$ crosses v, level in F_t returns the value representing the length of the suffix of $\text{path}_t(b_t)$ starting in v. Formally, the level function is defined in the following way.

Definition 11. *For $v \in W \cup B$ and $t \in \{0, \dots, n\}$ let $\text{level}_t(v) = \text{sec-dist}_t(v)$ if $v \in W$, $\text{level}_t(v) = \text{dist}_t(v)$ if $v \in B_t$, and $\text{level}_t(v) = 0$ otherwise.*

It may at first seem confusing that the level of a white vertex is the second maximum distance to a leaf. It is defined this way because, surprisingly, in every turn t the path $\text{path}_t(b_t)$ enters its white vertices through the edge determining the maximum distance from the white vertex to a leaf. We illustrate the introduced definitions in Fig. 3. We present there an example run of an online scenario together with changing levels of vertices. We mark the dispatching vertices in each turn. An important property of the level function is that the levels of vertices drop by at most one along both $\text{path}_t(b_t)$ and $\text{sec-path}_t(b_t)$.

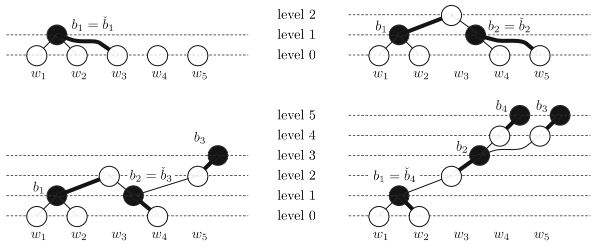


Fig. 3. Levels

Lemma 15 **(**Appendix C in [6]**).** *For* $v \in W \cup B_t$ *and* $u \in \{\mathrm{dir}_t(v), \text{sec-dir}_t(v)\}$ *it holds that* $|\mathrm{level}_t(v) - \mathrm{level}_t(u)| \leqslant 1$.

We are ready to move on to the proof of Theorem 1. We consider two cases:

1. the level of a dispatching vertex in turn t grows by at most a factor of β.
2. the level of a dispatching vertex in turn t grows by more than a factor of β.

where β is some constant value greater than 1 which we reveal later on. The total length of paths $\mathrm{path}_t^s(b_t)$ satisfying case (1) is bounded by Lemma 16 while the total length of paths $\mathrm{path}_t^s(b_t)$ satisfying case (2) is bounded by Lemma 17.

Lemma 16. *For cases when* $\mathrm{dist}_t(b_t) < \infty, \check{b}_t$ *is defined and* $\mathrm{level}_t(\check{b}_t) < \beta\,\mathrm{level}_{t-1}(\check{b}_t)$ *the total length of paths* $\mathrm{path}_t^s(b_t)$ *is bounded by* $2\beta n + \beta n \log_2 n$.

Proof. Let t be such that it satisfies the assumptions of the lemma. First observe that $b_t \neq \check{b}_t$, otherwise $\mathrm{level}_{t-1}(\check{b}_t) = 0 < \mathrm{level}_t(\check{b}_t)/\beta$. Due to Lemmas 8 and 9, b_t has precisely one neighbour alive in turn $t-1$.

In order to show an appropriate charging scheme, consider the final forest $F = F_n$. We study the connected components of a subforest $F[A_t]$ of F induced on vertices alive in turn t. Recall that vertices not yet presented are considered alive. In turn t some vertices, in particular b_t, die. Due to Lemma 10 vertices that die in turn t form a connected component D of $F[A_{t-1}]$. Then the connected component C of $\check{b}_t$ in $F[A_{t-1}]$ splits into D and components $C_1, \ldots, C_k$ in $F[A_t]$. Let C_1 be the largest component among $C_1, \ldots, C_k$. Due to Lemma 10, $\mathrm{path}_t^s(b_t)$ is contained entirely in one of the components $C_1, \ldots, C_k$, say $\mathrm{path}_t^s(b_t)$ is contained in C_i. If $i \neq 1$, we can charge the length of $\mathrm{path}_t^s(b_t)$ by charging 1 token to the vertices of C_i. A particular vertex can be charged at most $\log_2 n$ tokens this way, as each time it is charged its component halves the size. It remains to deal with the case when $\mathrm{path}_t^s(b_t)$ is contained in the largest component C_1. For the reference see Fig. 4. Let w^p be the predecessor of $\check{b}_t$ on $\mathrm{path}_t(b_t)$. Let T be the connected component of $\check{b}_t$ in F_{t-1} rooted at $\check{b}_t$ and let $T_\frown$ be the connected component of $\check{b}_t$ in F_{t-1} rooted at w^p. For the reference see Fig. 5. By Observation 14 and our assumptions it holds that $\|\mathrm{path}_t^s(b_t)\| = \mathrm{dist}_t(\check{b}_t) < \beta\,\mathrm{dist}_{t-1}(\check{b}_t) = \beta \min_{w \in \mathrm{Ch}_T(\check{b}_t)} \text{mini-max}_T(w) + \beta \leqslant \beta\,\text{mini-max}_T(w^p) + \beta$. The constant cost of β gives a total cost of βn over all turns. What remains to show is how to charge the cost of $\beta\,\text{mini-max}_T(w^p)$. Since $w^p \in A_{t-1}$ it holds that $\text{mini-max}_T(w^p) = \max_{b \in \mathrm{Ch}_{T_\frown}(w^p) \setminus \{\check{b}_t\}} \text{mini-max}_{T_\frown}(b) + 1 \leqslant \max_{b \in \mathrm{Ch}_{T_\frown}(w^p)} \text{mini-max}_{T_\frown}(b) + 1 = \mathrm{dist}_{t-1}(w^p) < \infty$. Thus $\text{mini-max}_T(w^p) = \|\text{mini-max-path}_T(w^p)\|$. We charge the vertices of $\text{mini-max-path}_T(w^p)$ to pay for the cost given by its length. Due to Lemma 7 it holds that $\text{mini-max-path}_T(w^p)$ visits only vertices that are alive in turn $t-1$. By definition $V(\text{mini-max-path}_T(w^p)) \cap C_1 = \emptyset$, so each vertex of $\text{mini-max-path}_T(w^p)$ either dies in turn t or is contained in C_i for $i > 1$. To pay for that, we charge β tokens to every vertex that dies in turn t and we charge β tokens to each vertex of components $C_2 \ldots C_k$. The total charge for this case sums up to $\beta n + \beta n \log_2 n$. If we add the charge we needed for other cases, we obtain a total of $2\beta n + \beta n \log_2 n$. $\square$

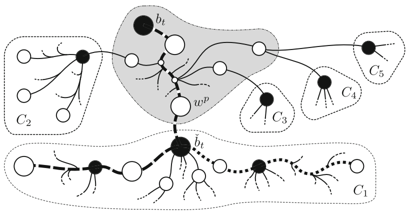


Fig. 4. Splitting the component C into $C_1, \ldots, C_5$ and D.

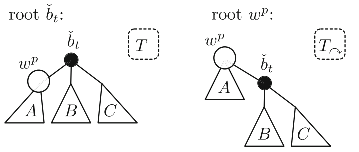


Fig. 5. The connected component of F_{t-1} rooted in $\check{b}_t$ and w^p.

Lemma 17. *For cases when* $\mathrm{dist}_t(b_t) < \infty$*,* $\check{b}_t$ *is defined and* $\mathrm{level}_t(\check{b}_t) \geqslant \beta\,\mathrm{level}_{t-1}(\check{b}_t)$ *the sum of the lengths of paths* $\mathrm{path}_t^s(b_t)$ *is bounded by* $\frac{\beta(\beta+1)}{(\beta-1)^2} n(2\ln n + 3.4) + n$*.*

Proof. Given the level function, we want to consider the vertices of $F = F_n$ whose level in turn t is above a certain value l. To be more precise, we need to consider the subforest of F induced by such vertices. This forest changes dynamically as the turns pass by. We describe it more formally below.

Definition 12. *For* $t \in [n]$ *and* $l \in \mathbb{N}$ *we define* $F_t^l = F[\{v \in W \cup B : l \leqslant \mathrm{level}_t(v)\}]$*.*

Recall that if $b \in B \setminus B_t$ then $\mathrm{level}_t(b) = 0$. For a subforest $F' = \langle V', E' \rangle$ of $F = \langle W \cup B, E \rangle$, we denote a connected component of vertex $v \in V'$ as $\mathrm{comp}(v, F')$. The family of all connected components is denoted as $\mathcal{C}(F') = \{\mathrm{comp}(v, F') : v \in V'\}$. For a fixed l we observe how F_t^l changes from turn $t-1$ to t. Since the level function is monotonic, i.e., it satisfies $\mathrm{level}_{t-1}(v) \leqslant \mathrm{level}_t(v)$ (see Observation 2), the following hold:

Observation 18. F_{t-1}^l *is a subforest of* F_t^l*. Also,* $V(F_t^l) = V(F_{t-1}^l) \cup \{v \in V(F) : \mathrm{level}_{t-1}(v) < l \leqslant \mathrm{level}_t(v)\}$*.*

We fix a turn t for which $\mathrm{dist}_t(b_t) < \infty$, $\check{b}_t$ is defined, and $\mathrm{level}_t(\check{b}_t) \geqslant \beta\,\mathrm{level}_{t-1}(\check{b}_t)$. The idea is the following. We let $\underline{l} = \mathrm{level}_{t-1}(\check{b}_t)$ and $\overline{l} = \mathrm{level}_t(\check{b}_t)$. For the purpose of the proof we need a function that describes some intermediate level between $\mathrm{level}_{t-1}()$ and $\mathrm{level}_t()$. We thus extend the level function to rational indices: $\mathrm{level}_{t-1/2}(v) = \mathrm{level}_{t-1}(v)$ if $v = \check{b}_t$ and $\mathrm{level}_{t-1/2}(v) = \mathrm{level}_t(v)$ otherwise. Observe that $\mathrm{level}_t(v)$ function is still monotonic in t after the extension. We illustrate these definitions by example in Fig. 6. We observe that on

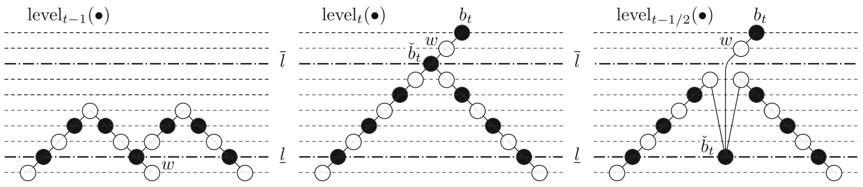


Fig. 6. Fractional levels

levels l from $\underline{l}$ to $\bar{l}-1$ the separate components of $F^l_{t-1/2}$ are merged in F^l_t. It is this merging that allows us to provide the charging scheme. The level defined for fractional indices may be interpreted as an additional fractional turn between $t-1$ and t.

Let us fix a level l. In every turn $t = 0, \frac{1}{2}, 1, 1\frac{1}{2}, \ldots, n$ a number of δ tokens is assigned to every connected component C in F^l_t such that $|C| \geqslant \rho l$, where δ and ρ are constants that we compute later. Smaller components are not assigned any tokens. Note that if l is large, only large components are assigned tokens. We plan to use these tokens to pay for the mini-max paths in each turn. First, however, we describe how we maintain such an assignment on level l.

First we consider moving from turn $t-1$ to turn $t-1/2$. The forest $F^l_{t-1/2}$ is obtained by adding the set of vertices $\Delta = V(F^l_{t-1/2}) \setminus V(F^l_{t-1}) = \{v \in B_t \cup W : \text{level}_{t-1}(v) < l \leqslant \text{level}_{t-1/2}(v)\}$ to F^l_{t-1} (see Observation 18). We want to add some structure to this process. We divide transformation from F^l_{t-1} to $F^l_{t-1/2}$ into two sub-phases. In the first sub-phase, the vertices of Δ form new singleton components: $\mathcal{C}' = \bigcup_{v\in\Delta}\{\langle\{v\},\emptyset\rangle\}$, where $\langle\{v\},\emptyset\rangle$ is a graph with only one vertex v and without edges. Together with the set of connected components of F^l_{t-1} (referred to as $\mathcal{C}(F^l_{t-1})$) they form a family $\mathcal{I} = \mathcal{C}(F^l_{t-1}) \cup \mathcal{C}'$. In the second sub-phase, components in $\mathcal{I}$ merge whenever there is an edge of F connecting them, finally becoming the connected components of $F^l_{t-1/2}$. Every component $C \in \mathcal{C}(F^l_{t-1/2})$ can be assigned a set of components $\mathcal{I}_C = \{C_1 \ldots C_k\} \subseteq \mathcal{I}$ that merged into C. There are two possible options:

(a) there is the component $C_i \in \mathcal{I}$ with size $|C_i| \geqslant \rho l$, so C_i is already assigned δ tokens
(b) every $C_i \in \mathcal{I}$ satisfies $|C_i| < \rho l$, so none of them is assigned any tokens.

In case (a), δ assigned to C_i, which ceases to exist, is now transferred to C. In case (b), if $|C| \geqslant \rho l$, every vertex $v \in C$ chips in with a payment of $\frac{\delta}{\rho l}$, so the vertices of C pay in total at least δ. We count the total amount that is paid at the end of the proof.

We now consider the transition from turn $t-1/2$ to turn t. There is only one vertex, mainly $\check{b}_t$, which changes its level. Its level increases from $\underline{l}$ to $\bar{l}$. Level l is only affected by this transition if $\underline{l} < l \leqslant \bar{l}$. So, the forest F^l_t is formed from $F^l_{t-1/2}$ by adding $\check{b}_t$. The only difference between $\mathcal{C}(F^l_{t-1/2})$ and $\mathcal{C}(F^l_t)$ is that some family of separate components of $\mathcal{C}(F^l_{t-1/2})$ becomes connected by $\check{b}_t$ and forms a new connected component $\text{comp}(\check{b}_t, F_t^l)$. The set of components that merge into $\text{comp}(\check{b}_t, F_t^l)$ is precisely $\mathcal{I}' = \mathcal{C}(\text{comp}(\check{b}_t, F_t^l) \setminus \check{b}_t) \cup \{\langle \check{b}_t, \emptyset \rangle\}$. The way of assigning δ to $\text{comp}(\check{b}_t, F_t^l)$ if $|\text{comp}(\check{b}_t, F_t^l)| \geqslant \rho l$ is the same as in the transition from $t-1$ to $t-1/2$. The difference is that now we want to utilize some of the assigned tokens to pay for the mini-max path in turn t. Thus, we distinguish three cases now:

(i) exactly one of the components $C' \in \mathcal{I}'$ satisfies $|C'| \geqslant \rho l$, so C' is assigned δ tokens,
(ii) every $C \in \mathcal{I}'$ satisfies $|C| < \rho l$ so none of them is assigned tokens,
(iii) two or more components $C', C'' \in \mathcal{I}'$ satisfy $|C'| \geqslant \rho l$ and $|C''| \geqslant \rho l$, so C' and C'' are both already assigned δ tokens.

Cases (i) and (ii) are handled in the exactly same manner as in the transition from $t-1$ to $t-1/2$. The difference is that in case (iii) we utilize δ tokens assigned to C' while δ tokens assigned to C'' transfer to $\text{comp}(\check{b}_t, F_t^l)$.

It remains to prove that the tokens utilized in turn t suffice to pay for $\|\text{path}_t^s(b_t)\|$. Observation 14 shows that $\text{level}_t(\check{b}_t) = \|\text{path}_t^s(b_t)\|$. So we need to pay $\text{level}_t(\check{b}_t)$ tokens when moving from turn $t-1/2$ to t. Let $\rho := (\beta-1)/(\beta+1)$. By Claim 19, proved later on, case (iii) occurs on at least $(\bar{l} - \underline{l})/2$ levels. Since $\underline{l} \leqslant \bar{l}/\beta$, we utilize at least $\delta(\bar{l} - \bar{l}/\beta)/2 = \frac{\delta(1-1/\beta)}{2} \text{level}_t(\check{b}_t)$ tokens. Setting $\delta := 2/(1-1/\beta)$ allows paying the desired amount.

Now we count the sum of lengths of $\text{path}_t^s(b_t)$. Every vertex pays $\frac{\delta}{\rho l}$ at most once per level and the highest level is not greater than $2n$, so the total amount paid by a vertex over all the turns is bounded by $\frac{\delta}{\rho}\sum_{l=1}^{2n}\frac{1}{l} \leqslant \frac{\delta}{\rho}(\ln(2n)+1)$. Hence, the total amount paid by all vertices is at most $\frac{\delta}{\rho}n(\ln n + 1.7)$. If we plug in the constants $\rho = (\beta-1)/(\beta+1)$ and $\delta = 2/(1-1/\beta)$ into above bound, we obtain that $\sum_{t\in\mathcal{T}} \|\text{path}_t^s(b_t)\| \leqslant \frac{\beta(\beta+1)}{(\beta-1)^2} n(2\ln n + 3.4)$. □

To complete the proof of Lemma 17 we move on to proving the following Claim.

Claim 19. *For a fixed* $t \in [n]$ *let* $l_0 = \text{level}_{t-1}(\check{b}_t) + 1$ *and* $l_1 = \lfloor(\text{level}_{t-1}(\check{b}_t) + \text{level}_t(\check{b}_t))/2\rfloor$. *For* $\rho = \frac{\beta-1}{\beta+1}$ *and* $l \in \{l_0, \ldots, l_1\}$ *there exist two different vertices* $w_1, w_2 \in \text{N}_t(\check{b}_t) \cap A_t$ *such that* $\text{comp}(w_1, F_{t-1/2}^l)$ *and* $\text{comp}(w_2, F_{t-1/2}^l)$ *are two separate components of* $F_{t-1/2}^l$ *and* $|\text{comp}(w_i, F_{t-1/2}^l)| \geqslant \rho l$ *for* $i \in \{1, 2\}$.

Proof. Fix $l \in \{l_0, \ldots, l_1\}$. By definition $\text{N}_t(\check{b}_t) \cap A_t \geqslant 2$. Thus, $\text{dist}_t(\check{b}_t) < \infty$ and $\text{sec-dist}_t(\check{b}_t) < \infty$. We show that $w_1 = \text{dir}_t(\check{b}_t)$ and $w_2 = \text{sec-dir}_t(\check{b}_t)$ satisfy the desired conditions.

First note that $w_1, w_2 \in V(F_{t-1/2}^l)$, because $\text{level}_t(w_i) \geqslant \text{level}_t(\check{b}_t) - 1 \geqslant l_1$ for $i \in \{1, 2\}$ due to Lemma 15. Note also that $\text{comp}(w_1, F_{t-1/2}^l)$ and $\text{comp}(w_2, F_{t-1/2}^l)$ are two separate components of $F_{t-1/2}^l$ because the only path connecting w_1 and w_2 in F is through $\check{b}_t$ and $\check{b}_t \notin V(F_t^l)$ because $\text{level}_{t-1/2}(\check{b}_t) = \text{level}_{t-1}(\check{b}_t) < l_0 \leqslant l$.

Due to Lemma 15 the levels of vertices drop by at most one along $\text{path}_t(\check{b}_t)$ and $\text{sec-path}_t(\check{b}_t)$. Let π_1 be the prefix of $\text{path}_t(\check{b}_t)$ of length $\bar{l} - l$ and π_2 be the prefix of $\text{sec-path}_t(\check{b}_t)$ of length $\bar{l} - l$, where $\bar{l} = \text{level}_t(\check{b}_t)$. It holds that if $v \in V(\pi_i)$ then $\text{level}_{t-1/2}(v) = \text{level}_t(v) \geqslant l$. Because $l_0 - 1 = \text{level}_{t-1}(\check{b}_t) \leqslant \text{level}_t(\check{b}_t)/\beta = \bar{l}/\beta$ and $l \leqslant l_1 = \lfloor (l_0 - 1 + \bar{l})/2 \rfloor$ we have $l \leqslant (\bar{l}/\beta + \bar{l})/2 = (1/\beta + 1) \cdot \bar{l}/2$. This implies $|\text{comp}(w, F_t^{\,l})| \geqslant \bar{l} - l = \frac{2\beta}{\beta+1} \cdot \frac{1+1/\beta}{2} \cdot \bar{l} - l \geqslant \frac{2\beta}{\beta+1} \cdot l - l = \frac{\beta-1}{\beta+1} \cdot l$. Setting $\rho = (\beta-1)/(\beta+1)$ completes the proof of the claim. □

We can now put all the pieces together to prove our main result.

Theorem 20. $\sum_{t\in[n]:\text{dist}_t(b_t)<\infty} \text{dist}_t(b_t) \in \mathcal{O}(n \log n)$.

Proof. By Definition 10 we have $\text{path}_t(b_t) = \text{path}_t^p(b_t) \cdot \text{path}_t^s(b_t)$. By Observation 13 it holds that $\sum_{t\in[n]:\text{dist}_t(b_t)<\infty} \|\text{path}_t^p(b_t)\| \leqslant 2n$. If $\check{b}_t$ is undefined, then $\text{path}_t^s(b_t)$ is empty so its length is 0. For the cases when $\check{b}_t$ is defined and $\text{level}_t(\check{b}_t) < \beta\,\text{level}_{t-1}(\check{b}_t)$ we have $\sum_{t\in[n]:\text{dist}_t(b_t)<\infty} \|\text{path}_t^s(b_t)\| \leqslant 2\beta n + \beta n \log n$. For the cases when $\check{b}_t$ is defined and $\text{level}_t(\check{b}_t) \geqslant \beta\,\text{level}_{t-1}(\check{b}_t)$ we have $\sum_{t\in[n]:\text{dist}_t(b_t)<\infty} \|\text{path}_t^s(b_t)\| \leqslant \frac{\beta(\beta+1)}{(\beta-1)^2} n(2\ln n + 3.4) + n$. This gives the theorem statement for any $\beta > 1$. □

References

1. Bernstein, A., Holm, J., Rotenberg, E.: Online bipartite matching with amortized $O(\log^2 n)$ replacements. arXiv:1707.06063 (2017)
2. Bernstein, A., Stein, C.: Fully dynamic matching in bipartite graphs. In: ICALP, Part I, pp. 167–179 (2015)
3. Birnbaum, B.E., Mathieu, C.: On-line bipartite matching made simple. SIGACT News **39**(1), 80–87 (2008)
4. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Online bipartite matching in oine time. In: FOCS, pp. 384–393 (2014)
5. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Shortest augmenting paths for online matchings on trees. In: WAOA, pp. 59–71 (2015)
6. Bosek, B., Leniowski, D., Sankowski, P., Zych-Pawlewicz, A.: A tight bound for shortest augmenting paths on trees. arXiv:1704.02093v2 (2017)
7. Chaudhuri, K., Daskalakis, C., Kleinberg, R.D., Lin, H.: Online bipartite perfect matching with augmentations. In: INFOCOM, pp. 1044–1052 (2009)
8. Devanur, N.R., Jain, K., Kleinberg, R.D.: Randomized primal-dual analysis of RANKING for online bipartite matching. In: SODA, pp. 101–107 (2013)
9. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972)
10. Grove, E.F., Kao, M.-Y., Krishnan, P., Vitter, J.S.: Online perfect matching and mobile computing. In: Akl, S.G., Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp. 194–205. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60220-8_62
11. Gupta, A., Kumar, A., Stein, C.: Maintaining assignments online: matching, scheduling, and flows. In: SODA, pp. 468–479 (2014)

12. Gupta, M., Peng, R.: Fully dynamic (1+ε)-approximate matchings. In: FOCS, pp. 548–557 (2013)
13. Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for on-line bipartite matching. In: STOC, pp. 352–358 (1990)
14. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: STOC, pp. 745–754 (2013)
15. Sankowski, P.: Faster dynamic matchings and vertex connectivity. In: SODA, pp. 118–126 (2007)

Approximation Algorithms for Replenishment Problems with Fixed Turnover Times

Thomas Bosman[1(✉)], Martijn van Ee[6], Yang Jiao[2], Alberto Marchetti-Spaccamela[3,5], R. Ravi[2], and Leen Stougie[1,4,5]

[1] Vrije Universiteit, Amsterdam, The Netherlands
{thomas.bosman,l.stougie}@vu.nl
[2] Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
{yangjiao,ravi}@andrew.cmu.edu
[3] Sapienza University of Rome, Rome, Italy
alberto@dis.uniroma1.it
[4] Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands
stougie@cwi.nl
[5] Erable, Inria, Paris, France
[6] Netherlands Defence Academy, Den Helder, The Netherlands
m.v.ee.01@mindef.nl

Abstract. We introduce and study a class of optimization problems we coin replenishment problems with fixed turnover times: a very natural model that has received little attention in the literature. Nodes with capacity for storing a certain commodity are located at various places; at each node the commodity depletes within a certain time, the turnover time, which is constant but can vary between locations. Nodes should never run empty, and to prevent this we may schedule nodes for replenishment every day. The natural feature that makes this problem interesting is that we may schedule a replenishment (well) before a node becomes empty, but then the next replenishment will be due earlier also. This added workload needs to be balanced against the cost of routing vehicles to do the replenishments. In this paper, we focus on the aspect of minimizing routing costs. However, the framework of recurring tasks, in which the next job of a task must be done within a fixed amount of time after the previous one is much more general and gives an adequate model for many practical situations.

Note that our problem has an infinite time horizon. However, it can be fully characterized by a compact input, containing only the location of each store and a turnover time. This makes determining its computational complexity highly challenging and indeed it remains essentially unresolved. We study the problem for two objectives: MIN-AVG minimizes the average tour length and MIN-MAX minimizes the maximum tour length over all days. For MIN-MAX we derive a logarithmic factor approximation for the problem on general metrics and a 6-approximation for the problem on trees, for which we have a proof of NP-hardness. For MIN-AVG we present a logarithmic approximation on general metrics, 2-approximation for trees, and a pseudopolynomial time algorithm for the line. Many intriguing problems remain open.

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 217–230, 2018.
https://doi.org/10.1007/978-3-319-77404-6_17

1 Introduction

Imagine the following particular inventory-routing problem. A set of automatic vendor machines are spread over a country or a city. They have a certain turnover time: the number of days in which a full machine will be sold out. Replenishment is done by vehicles. Let us assume that turnover times are machine dependent but not time dependent, and that it is highly undesirable to have an empty machine. However, the holding costs of the machine are negligible, so that we will always fill the machine to capacity. There is nothing against replenishing a machine before it has become empty, but then the next replenishment will due earlier as well. That is, the deadline of the next replenishment is always within the turnover time after the last replenishment. Equivalently, in any consecutive number of days equal to the turnover time, at least one replenishment has to take place. Replenishing a machine earlier to combine it with the replenishment of another machine that is due earlier may lead to cost savings. The feature that makes this problem so special w.r.t. existing literature, is that it can be compactly modeled by only specifying for every machine its location and the turnover time. The feature is very natural but has hardly been studied in the existing literature. There are intriguing basic open complexity questions, and some highly non-trivial results.

The motivation for studying this problem comes linea recta from a business project for the replenishment of ATMs in the Netherlands, in which some of the co-authors are involved. The replenishment of the ATMs of all the large banks in the Netherlands has been outsourced to a single company: Geld Service Nederland. Of course the real-life ATM replenishment problem is not as stylized as described above; the turnover time is not strictly the same over time but subject to variability, there are restrictions on the routes for the vehicles, etc. But the feature that is least understood in the ATM-problem is exactly the problem of how to deal with the trade-off between replenishing an ATM earlier than its due date leading to a higher frequency of replenishments and the savings on vehicle routing costs.

Formally, an instance of the problem that we study in this paper, which we baptize the REPLENISHMENT PROBLEM WITH FIXED TURNOVER TIMES (RFTT), consists of a pair (G, τ), where $G = (V \cup \{s\}, E, c)$ is a weighted graph with a designated depot vertex s and weights on the edges $c : E \rightarrow \mathbb{R}_+$, and turnover times $\tau \in \mathbb{N}^{|V|}$, indicating that $v_j \in V$ should be visited at least once in every interval of τ_j days.

A solution consists, for each day k, of a tour T_k in G starting in and returning to the depot s and visiting a subset of the vertices $J_k \subseteq V$. It is feasible if $v_j \in \bigcup_{k=t+1}^{t+\tau_j} J_k$, $\forall t$ and $\forall v_j \in V$. We will focus on solutions that repeat themselves after a finite amount of time, that is, in which $(T_k, \ldots, T_{k+\ell}) = (T_{k+\ell+1}, \ldots, T_{k+2\ell})$ for some ℓ, and all k. Since all turnover times are finite, this is no real restriction.

We consider two versions of RFTT. In the first version, called MIN-AVG, the goal is to find a feasible solution that minimizes the average tour length. In the MIN-MAX problem, we want to find a feasible solution that minimizes the maximum tour length over all days.

We emphasize that the particular feature of this model, that jobs or visits to clients recur and need to be done within each job-specific consecutive time interval occurs naturally in many problem settings. It allows any job of a recurring task to be done before its deadline, but then the next job of the task comes earlier and hence its deadline. This is a feature that, despite its natural applicability, has hardly been studied in the literature from a theoretical point of view.

Related work. As mentioned before, our problem can be seen as a special case of the Inventory Routing Problem (IRP) [8]. Here, clients (vertices) have their own storage with a certain capacity and for each day a demand is specified. The clients pay holding cost over their inventory. However, omitting inventory cost, we can interpret our problem as such an inventory routing problem in which the demand at any given location is the same every day, leading to a very small input description of our problem consisting only of a location and a turnover time (storage capacity divided by daily demand), which makes it incomparable to the inventory routing problem from a complexity point of view. Indeed it is unclear if the decision version of our problem is in NP or in co-NP.

Another closely related problem is the Periodic Latency Problem [9], which features the recurring visits requirement of rftt. We are given recurrence length q_i for each client i and travel distances between clients. Client i is considered *served* if it is visited every q_i time units. The server does not return to the depot at the end of each time unit (e.g. day), but keeps moving continuously between clients at uniform speed. Another difference between Periodic Latency Problem and rftt is the objective function. Coene et al. [9] study two versions of the problem: one that maximizes the number of served clients by one server, and one that minimizes the number of servers needed to serve all clients. They resolve the complexity of these problems on lines, circles, stars, trees, and general metrics.

A problem that does share the compact input size and is in fact a very special case of our problem is known under the guise of Pinwheel Scheduling. It has been introduced to model the scheduling of a ground station to receive information from a set of satellites without data loss. In terms of our problem no more than one vertex can be replenished per day and all distances to the depot are the same; the interesting question here is if there exists a feasible schedule for replenishing the vertices. Formally, a set of jobs $\{1, \ldots, n\}$ with periods $p_1, \ldots, p_n$ is given, and the question is whether there exists a schedule $\sigma : \mathbb{N} \to \{1, \ldots, n\}$ such that $j \in \bigcup_{k=t+1}^{t+p_j} \sigma_k$, $\forall t \geq 0$ and $\forall j$.

Pinwheel Scheduling was introduced by Holte et al. [17], who showed that it is contained in PSPACE. The problem is in NP if the schedule σ is restricted to one in which for each job the time between two consecutive executions remains constant throughout the schedule. In particular this holds for instances with density $\rho = \sum_j 1/p_j = 1$ [17]. They also observed that the problem is easily solvable when $\rho \leq 1$ and the periods are harmonic, i.e. p_i is a divisor of p_j or vice versa for all i and j. As a corollary, every instance with $\rho \leq \frac{1}{2}$ is feasible.

Chan and Chin [7] improved the latter result by giving an algorithm that produces a feasible schedule for Pinwheel Scheduling whenever $\rho \leq \frac{2}{3}$. In [6], they improved this factor to $\frac{7}{10}$. Later, Fishburn and Lagarias [14] showed that every instance with $\rho \leq \frac{3}{4}$ has a feasible schedule. All these papers work towards the conjecture that there is a feasible schedule if $\rho \leq \frac{5}{6}$. That this bound is tight can be seen by the instance with $p_1 = 2$, $p_2 = 3$ and $p_3 = M$, with M large. This instance cannot be scheduled, but has a density of $\frac{5}{6} + \frac{1}{M}$.

The complexity of Pinwheel Scheduling has been open since it was introduced. It was only recently shown by Jacobs and Longo [18] that there is no pseudopolynomial time algorithm solving the problem unless SAT has an exact algorithm running in expected time $n^{O(\log n \log \log n)}$, implying for example that the randomized exponential time hypothesis fails to hold [5,10]. Since the latter is unlikely, one could conclude that Pinwheel Scheduling is not solvable in pseudopolynomial time. It remains open whether the problem is PSPACE-complete.

Similar to Pinwheel Scheduling, the k-server Periodic Maintenance Problem [2,11,19] has n jobs, each with a specified periodicity and a processing time. Each server may serve at most one job per time unit. However, job i is required to be served exactly every m_i days apart rather than within every m_i days. The case $k = 1, c_j = 1$ for all j is analogous to Pinwheel Scheduling, except for the exact periodicity constraint. For any $k \geq 1$, Mok et al. [19] have shown it is NP-complete in the strong sense. For the special case when m_i are multiples of each other or when there are at most 2 different periodicities, they have shown it is in P.

Other related problems with a compact input representation include real-time scheduling of sporadic tasks [1,3], where we are given a set of recurrent tasks. On a single machine, EDF (Earliest Deadline First) is optimal. However, we remark that the complexity of deciding whether a given set of tasks is feasible has been open for a long time and only recently proved showing that it is coNP-hard to decide whether a task system is feasible on a single processor even if the utilization is bounded [12].

Another related problem is the Bamboo Garden Trimming Problem introduced by Gasieniec et al. [16]. There are n bamboos, each having a given growth rate, which may be viewed as inducing a periodicity. On each day, a robot may trim at most one bamboo back to height 0. The goal is to minimize the maximum height of the bamboos. Gasieniec et al. provide a 4-approximation for the general case and a 2-approximation for balanced growth rates.

This paper. We investigate the computational complexity of both the min-max and the min-avg version of rftt. Mostly we will relate their complexity to the complexity of Pinwheel Scheduling. Some interesting inapproximability results follow from this relation. After that, we will start with some special cases. In Sect. 3, we give our most remarkable result, a constant factor approximation for min-max on a tree, next to a less remarkable constant approximation for the min-avg version on the tree. In the same section, we show for min-avg that the problem can be solved to optimality in pseudopolynomial time on line metrics. Finally, in Sect. 4, we present logarithmic factor approximations for both problem versions on general metrics.

2 Complexity

In this section, we investigate the computational complexity for both objectives. Since our problem requires finding a shortest tour visiting some subset of vertices for every day, it is at least as hard as the TRAVELING SALESMAN PROBLEM (TSP). However it is also interesting to note that the problems are at least as hard as PINWHEEL SCHEDULING as well. For the MIN-MAX objective there is a direct reduction showing that a factor 2 approximation is at least as hard as PINWHEEL SCHEDULING: construct an unweighted star with the depot at the center and each leaf corresponding to a job in the pinwheel instance. This instance has value 2 only if there exists a pinwheel schedule and at least 4 otherwise.

For the MIN-AVG RFTT the reduction is a bit more involved, and given in the appendix of the full version of this paper [4].

Theorem 1. *On series-parallel graphs,* MIN-AVG RFTT *is at least as hard as* PINWHEEL SCHEDULING.

We note that this hardness result is incomparable to the TSP reduction. Pinwheel is neither known to be NP-hard nor in NP, although it is conjectured to be PSPACE-complete.

Lastly, as Theorem 2 shows, the MIN-MAX RFTT remains hard even on star graphs (where TSP is trivial). A reduction can be found in the appendix of the full paper [4].

Theorem 2. MIN-MAX RFTT *is NP-hard on star graphs.*

3 Approximation on Trees

In this section we give a 2-approximation for MIN-AVG and a 6-approximation for MIN-MAX on trees.

We start out with a simplifying result, which will also be of use in the next sections.

Lemma 1. *Given an instance* (G, τ) *of* RFTT, *let* τ' *be found by rounding every turnover time in* τ *down to a power of* 2. *Then* $OPT(G, \tau') \leq 2OPT(G, \tau)$ *for both* MIN-AVG *and* MIN-MAX *objectives.*

Proof. Let $(G, \bar{\tau})$ denote the instance found from (G, τ) by rounding every turnover time up to a power of 2. Since any schedule remains feasible if we round up the turnover times, we have that $OPT(G, \bar{\tau}) \leq OPT(G, \tau) \leq OPT(G, \tau')$.

Suppose we have an optimal solution for $(G, \bar{\tau})$ in which $\bar{T}_k$ is scheduled on day k. We can construct a feasible schedule for (G, τ') by scheduling the concatenation of $\bar{T}_{2k-1}$ and $\bar{T}_{2k}$ on day k. The maximum tour length in this schedule is at most twice that of the optimal solution for $(G, \bar{\tau})$ and every tour from the original schedule is visited exactly twice in the new schedule, so this yields a factor 2 increase in both the MIN-MAX and the MIN-AVG objective.

In the remainder we assume w.l.o.g. that G is rooted at s and that turnover times are increasing on any path from the depot to a leaf node in G. Furthermore, for an edge e in E we define $D(e)$ to be the set of vertices that are descendants of e. We also need the following definition.

Definition 1 (tt-weight of an edge). *For any edge in G we define:*

$$q(e) = \min_{j \in D(e)} \tau_j.$$

We call this quantity the tt-weight (turnover time-weight) of e.

This definition allows us to express the lowerbound in Lemma 2.

Lemma 2 (tt-weighted tree). *For an instance (G, τ) of the* RFTT *on trees it holds that the average tour length is at least:*

$$L(G, \tau) := 2 \sum_{e \in E} \frac{c(e)}{q(e)}.$$

Proof. This follows immediately from the fact that $\frac{2}{q(e)}$ lower bounds the number of times edge e must be traversed on *average* in any feasible solution.

Since the maximum tour length is at least the average tour length, Lemma 2 also provides a lower bound for the MIN-MAX objective.

An approximation for MIN-AVG RFTT is thus found by rounding all turnover times to powers of 2 and then visit each client j on every day that is a multiple of τ_j. Since in that case the lower bound of Lemma 2 is exactly attained on the rounded instance, Lemma 1 implies the following theorem.

Theorem 3. *There is a 2-approximation for* MIN-AVG RFTT *on trees.*

3.1 MIN-MAX

We will now show that we can achieve a 6-approximation for MIN-MAX RFTT on trees by providing a 3-approximation algorithm if all turnover times are powers of 2 and then applying Lemma 1.

The main idea is to take a TSP-tour and recursively split it to obtain a schedule for the clients with increasing turnover times. During the splitting process, we assign each client j on that tour to a congruence class $\bar{a}_{\tau_j} = \{k \in \mathbb{N} | k \equiv a \pmod{\tau_j}\}$ for some $a \leq \tau_j$, to indicate we want to visit j on each day in $\bar{a}_{\tau_j}$. Similarly, we distribute all edges e to a congruence class $\bar{a}_{q(e)}$. We do this ensuring that on any given day, we can create a tour through all clients associated with that day, using the edges associated with that day plus a small set of extra edges.

Let us define some further notation. For a given congruence class $\bar{a}_m \subseteq \mathbb{N}$, we denote $g(\bar{a}_m) \subseteq V$ the set of vertices and $f(\bar{a}_m) \subseteq E$ the set of edges assigned to that class. Note that $\bar{a}_m$ and $(\overline{a+m})_m$ define the same congruence class, so

$f(\bar{a}_m) = f((\overline{a+m})_m)$. Then, for any $k \in \mathbb{N}$ we have that J_k, the set of clients we need to visit on day k, is

$$J_k = \bigcup_{m\in\mathbb{N}, a\leq m | k \in \bar{a}_m} g(\bar{a}_m).$$

Algorithm 1. Algorithm for recursively constructing $f(\cdot)$ and $g(\cdot)$

```
  function RecurseTreeSchedule($\mathbf{d}, a, m$)
Require: $\mathbf{d}$, a connected sequence of edges in $G$, powers of 2 turnover times $\tau$; $a, m$,
  integers
      if $\mathbf{d} \neq \emptyset$ then
          $f(\bar{a}_m) = \{e \in \mathbf{d} \mid q(e) = m\}$
          $g(\bar{a}_m) = \{j \in V(\mathbf{d}) \mid \tau_j = m\}$
          $k = \max_{k'}$ s.t. $\sum_{i\in[k'-1]|q(d_i)>m} \frac{c(d_i)}{q(d_i)} \leq \frac{1}{2} \sum_{i\in[n]|q(d_i)>m} \frac{c(d_i)}{q(d_i)}$
          $\mathbf{d}^1 = (d_1, \ldots, d_{k-1})$
          $\mathbf{d}^2 = (d_{k+1}, \ldots, d_n)$
          RecurseTreeSchedule($\mathbf{d}^1, a, 2m$), RecurseTreeSchedule($\mathbf{d}^2, a+m, 2m$)
      end if
  end function
```

The assignment of vertices and edges to classes is guided by the recursive splitting of a TSP-tour in G. The full procedure for constructing $f(\cdot)$ and $g(\cdot)$ is shown in Algorithm 1. The algorithm is initially called with $\mathbf{d}$, a TSP-tour visiting all vertices in G, and $a = m = 1$ and will determine the set of vertices to be visited on every day (i.e., those congruent to $\bar{a}_1$). Then the first (second) recursive call determines the sets of vertices with turnover time 2 that will be visited on odd (even) days. Analogously, RecurseTreeSchedule($\mathbf{d}^1, a, m$) will return the set of vertices with turnover time m to be visited on days in the congruence class $\bar{a}_m$ and the two recursive calls will return the set of vertices with turnover time $2m$ that are visited on days $a, a+2m, a+4m, \ldots$ and $a+m$, $a+3m, a+5m, \ldots$, respectively.

In the remainder we assume that any call to $f(\cdot)$ and $g(\cdot)$ returns the empty set for any argument that is not explicitly handled in Algorithm 1. Note that we use the notation $V(A)$ to denote the vertices incident to edges in $A \subseteq E$.

Lemma 3. *After Algorithm 1 terminates, each vertex j appears in some set $g(\bar{a}_{\tau_j})$ for some a.*

Proof. Note that $\mathbf{d^1} \cap \mathbf{d^2} = \emptyset$ and that $|\mathbf{d^1} \cup \mathbf{d^2}| = |\mathbf{d}| - 1$; since $\mathbf{d}$ is a connected set of edges then in each call to RecurseTreeSchedule, $V(\mathbf{d}^1) \cup V(\mathbf{d^2}) = V(\mathbf{d})$. Therefore no vertex is skipped in the construction of $g(\cdot)$.

In order to find a tour on day k through the vertices in J_k we use edges in $\bigcup_{h=1,2,\ldots,m} f(\bar{a}_h)$; as we already observed this set of edges does not necessarily connect vertices in $g(\bar{a}_m)$ to the depot. The next lemma shows that a tree that connects all vertices in $g(\bar{a}_m)$ to the root can be found by considering $\cup_{h=1,2,\ldots,m} f(\bar{a}_h)$ and adding a shortest path from some vertex in $g(\bar{a}_m)$ to the depot.

Lemma 4. *Let a, m be such that $f(\bar{a}_m)$ is nonempty. Let P be the set of edges on the shortest path connecting some arbitrary edge in $f(\bar{a}_m)$ to the root of G. Then the following edge set forms a connected component:*

$$T(\bar{a}_m) := P \cup \big(\bigcup_{h=1,2,\ldots,m} f(\bar{a}_h) \big).$$

Moreover, $T(\bar{a}_m)$ spans $\bigcup_{h=1,2,\ldots,m/2,m} g(\bar{a}_h)$.

Proof. To prove our first claim, we first show that for $k \leq m$, $f(\bar{a}_k)$ either induces at most one connected component, or each component it induces is incident to a component induced by $\bigcup_{h=1,2,\ldots,k/2} f(\bar{a}_h)$. Then, we will show that if $f(\bar{a}_k)$ induces at most one connected component, it is incident to $P \cup (\bigcup_{h=1,2,\ldots,k/2} f(\bar{a}_h))$.

Suppose $f(\bar{a}_k)$ does not induce at most one component. Note that $f(\bar{a}_k)$ is the subset of edges in some connected edge sequence $\mathbf{d}$ through G that have tt-weight k. But by the way tt-weight is defined and the fact that G is a tree, a simple path connecting disjoint edges with tt-weight k, can only consist of edges with tt-weight at most k. So every two components in $f(\bar{a}_k)$ are connected through a path of edges with tt-weight of at most k. Moreover since the sequence $\mathbf{d}$ used to construct $f(\bar{a}_k)$ is a subset of the sequence used to construct $f(\bar{a}_{k/2})$, by induction these connecting paths must be contained in $\bigcup_{h=1,2,\ldots,k/2} f(\bar{a}_h)$, as required.

Next we show that for any $k \leq m$ such that $f(\bar{a}_k) \neq \emptyset$, if $f(\bar{a}_k)$ is not incident to P then it is incident to $\bigcup_{h=1,\ldots,k/2} f(\bar{a}_h)$.

Let $\mathbf{d}$ be the sequence that was used to construct $f(\bar{a}_k)$. Since $\mathbf{d}$ contains all edges in $f(\bar{a}_m)$ and P contains at least one such edge, there exists a minimal path Q that contains some edge e in $f(\bar{a}_k)$ such that Q is connected to P. Moreover since Q is minimal and P contains the root, e must be the edge furthest away from the root on Q. This implies that all edges on Q have tt-weight k or less. Now suppose that Q contains edges with tt-weight strictly less than k. Then those edges are necessarily in $\bigcup_{h=1,\ldots,k/2} f(\bar{a}_h)$ and therefore $f(\bar{a}_k)$ is incident to that set. If not then Q is strictly contained in $f(\bar{a}_k)$ and therefore $f(\bar{a}_k)$ is connected to P.

The first claim of our lemma now follows by induction. $P \cup f(\bar{a}_1)$ is clearly connected. If $P \cup \bigcup_{h=1,2,\ldots,k/2} f(\bar{a}_h)$ is connected, we get that $f(\bar{a}_k)$ is either empty or is connected to P or to $\bigcup_{h=1,2,\ldots,k/2} f(\bar{a}_h)$, and the result follows.

To prove our second claim, suppose that for some k and $j \in g(\bar{a}_k)$ it holds that no edge incident to j, is in $\bigcup_{h=1,2,\ldots,k} f(\bar{a}_h)$. We will show that j appears on P, from which our claim immediately follows.

Let $\mathbf{d}$ be the sequence used to construct $g(\bar{a}_k)$. The edge e incident to j that is closest to the root, satisfies $q(e) \leq k$. So, it cannot be in $\mathbf{d}$ otherwise it would be contained in $\bigcup_{h=1,2,\ldots,k} f(\bar{a}_h)$. But this implies that e cuts off every edge in $\mathbf{d}$ from the root, and therefore e appears on P, as claimed, concluding the proof.

The next lemma allows us to bound the cost of edges included in $f(\bar{a}_h)$.

Lemma 5. *During each (recursive) call to* RecurseTreeSchedule*, it holds that*

$$\sum_{e\in\mathbf{d}|q(e)\geq m} m\frac{c(e)}{q(e)} + \sum_{h=1}^{m/2} \sum_{e\in f(\bar{a}_h)|q(e)=h} c(e) \leq L(G,\tau).$$

Proof. The proof is by induction on m. Since we initially call the algorithm with $\mathbf{d}$ a TSP-tour in G, which visits each edge twice, it clearly holds for $m = 1$.

Now for $m > 1$, suppose it holds for all smaller m. Without loss of generality, suppose we have a call to the function with input $\mathbf{d}^1, a, m$, such that $\mathbf{d}, a, m/2$ are the input parameters for its parent in the call stack.

$$\begin{aligned}
&\sum_{e\in\mathbf{d}^1|q(e)\geq m} m\frac{c(e)}{q(e)} + \sum_{h=1,2,\ldots,\frac{m}{2}} \sum_{e\in f(\bar{a}_h)|q(e)=h} c(e)\\
=&\sum_{e\in\mathbf{d}^1|q(e)\geq m} m\frac{c(e)}{q(e)} + \sum_{e\in f(\bar{a}_{m/2})|q(e)=m/2} \frac{m}{2}\frac{c(e)}{q(e)} + \sum_{h=1,2,\ldots,\frac{m}{4}} \sum_{e\in f(\bar{a}_h)|q(e)=h} c(e)\\
\leq&\sum_{e\in\mathbf{d}|q(e)\geq m} \frac{m}{2}\frac{c(e)}{q(e)} + \sum_{e\in f(\bar{a}_{m/2})|q(e)=m/2} \frac{m}{2}\frac{c(e)}{q(e)} + \sum_{h=1,2,\ldots,\frac{m}{4}} \sum_{e\in f(\bar{a}_h)|q(e)=h} c(e)\\
\leq&\sum_{e\in\mathbf{d}|q(e)\geq m/2} \frac{m}{2}\frac{c(e)}{q(e)} + \sum_{h=1,2,\ldots,\frac{m}{4}} \sum_{e\in f(\bar{a}_h)|q(e)=h} c(e) \leq L(G,\tau)
\end{aligned}$$

For the first equality, we split the second sum into an $h = m/2$ part and an $h = 1,\ldots,m/4$ part. In the first inequality we used the way $\mathbf{d}^1$ and $\mathbf{d}^2$ are determined in Algorithm 1, in the second inequality we used that $f(\bar{a}_m) \subseteq \mathbf{d}$ and in the last inequality we used the inductive hypothesis, concluding the proof.

We are now ready for the main theorem.

Theorem 4. *There is a* 6*-approximation for* min-max rftt *on trees.*

Proof. We first round all turnover times down to powers of 2, which loses a factor of 2 in the optimal solution. We then use Algorithm 1 to construct $f(\cdot)$ and $g(\cdot)$ thus determining the set of vertices J_k to be visited on day k. By Lemma 3 this defines a feasible schedule.

If we then take $T(\bar{k}_{\tau_{max}})$ as in Lemma 4, we get a tree that spans J_k. Moreover the weight of $T(\bar{k}_{\tau_{max}})$ is at most $\frac{3}{2}OPT$: the contribution of P is at most $\frac{1}{2}OPT$, since we need to reach any client at least on some day (and drive back), while the contribution of $\bigcup_{h=1,\ldots\tau_{max}} f(\bar{k}_h)$ is at most $L(G,\tau) \leq OPT$, which can be seen by applying Lemma 5 for $m = 2\tau_{max}$. Lastly, since we need a tour around $T(\bar{k}_{\tau_{max}})$, we lose another factor 2. This gives the approximation factor of 6.

It remains to show that Algorithm 1 runs in polynomial time, and that we can find a polynomial representation for the schedule. For the first claim, note that in each recursive call to the algorithm, the following equality holds $|\mathbf{d^1} \cup \mathbf{d^2}| = |\mathbf{d}| - 1$; hence the algorithm terminates after at most $2|E(G)|$ calls.

For the second claim, the crucial observation is that we only need to store the entries of g for a and m such that $g(\bar{a}_m)$ is nonempty. Since at most one entry is defined in every call to the algorithm, and we can simply check if $k \equiv a \pmod{m}$ for all stored entries, the claim, and the theorem, follow.

3.2 MIN-AVG on the Line

As an even more special underlying metric, we might consider the MIN-AVG problem on the line (on a path). For the MIN-MAX version this case is trivial, but for the MIN-AVG version its complexity is unclear: we do not know whether it is in NP, although we expect it to be NP-hard.

On the positive side we can show that the problem is not strongly NP-hard.

Theorem 5. MIN-AVG *on the line can be solved in pseudopolynomial time.*

We give a DP that finds an optimal schedule in polynomial time for any instance with polynomially bounded turnover times. Since we are minimizing the average, it is easy to see that we can reduce this problem to two times the MIN-AVG problem on the half-line (a path with the depot in one of the leaves). On the half-line each vertex i has a distance $d_i \in \mathbb{N}$ from the origin. Suppose vertices are numbered such that $d_1 \leq \ldots \leq d_n$. We present a pseudopolynomial time dynamic programming agorithm for this problem, based on the following observations.

First of all, we note that on any tour visiting vertex j automatically visits every vertex $i < j$. As in the tree case, we therefore assume that $\tau_i \leq \tau_j$ for $i < j$. Thus, after visiting j, all $i \leq j$ have a remaining turnover time of τ_i. For the dynamic program to work, we guess L, the day on which vertex n is visited for the first time and try all guesses between 1 and τ_n.

The dynamic program now works as follows. Suppose we are given the optimal solution for vertices $1, \ldots, i-1$ when only considering the days $1, \ldots, k$. Now we want to include i in the optimal solution for the first k days. If $k < \min\{\tau_i, L\}$, it is not necessary to visit i during the first k days, and hence it is optimal to take the optimal solution for the first $i-1$ vertices and k days. Otherwise, we need to visit i on some day ℓ in $\{1, \ldots, \min\{\tau_i, L\}\}$. Before day ℓ, we only need to visit the vertices $1, \ldots, i-1$. Thus, we take the optimal $\ell - 1$ tours for visiting the first $i-1$ vertices in the first $\ell - 1$ days. After day ℓ, all vertices have the same remaining turnover time as they had at time zero. Hence, we can take the optimal tours for the first i vertices and $k - \ell$ days.

Let $\phi_L(i,k)$:= the minimum cost of the first k tours visiting vertices $1, \ldots, i$. We initialize $\phi_L(0,k) = \phi_L(i,0) = 0$ and we use the recursion:

$$\phi_L(i,k) = \begin{cases} \phi_L(i-1,k), & \text{if } k < \min\{\tau_i, L\} \\ \displaystyle\min_{\ell=1,\ldots,\min\{\tau_i,L\}} \phi_L(i-1,\ell-1) + d_i + \phi_L(i,k-\ell), & \text{else} \end{cases}$$

The optimal solution is the schedule that corresponds to the value $L \in \{1, \ldots, \tau_n\}$ minimizing $\phi_L(n, L)/L$. Note that the algorithm runs in time $O(n\tau_n^3)$, implying the theorem.

4 Approximation on General Graphs

We will now present logarithmic approximations for both objectives. Note that an $O(\log \tau_{max})$-approximation is readily achieved; simply treat the sets of clients with equal turnover time as independent instances. For MIN-AVG, the problem with equal turnover times is simply TSP, for the MIN-MAX we get a problem sometimes called the k-TSP, for which a $\frac{5}{2}$ approximation is known [15]. Since by rounding to powers of 2, we ensure there are $O(\log(\tau_{max}))$ different turnover times, we get Theorem 6.

Theorem 6. MIN-MAX *and* MIN-AVG RFTT *have an* $O(\log \tau_{max})$*-approximation.*

Proof. By Lemma 1 we may assume every τ_i is a power of 2 so that there are at most $\log \tau_{max}$ different turnover times. We simply treat the sets of vertices with the same turnover time as separate instances and concatenate the solutions. Our result then follows from the fact that for all these instances a constant factor approximation is available. In the case of the MIN-MAX objective we get the k-TSP problem, where k is equal to the turnover time of the vertices in the instance. In the case of MIN-AVG, we need to minimize the sum over all k tours. But since all turnover times are equal there is no advantage to visiting vertices on different days, hence we recover a simple TSP problem.

In the case of MIN-MAX it is relatively simple to adapt this idea for an $O(\log n)$-approximation by appropriately reassigning clients to lower turnover times, as per Theorem 7.

Theorem 7. MIN-MAX RFTT *has an* $O(\log n)$*-approximation.*

Proof. We start by assuming that every turnover time is a power of 2. Next, we split up the instance into two new instances. To this end we first define a turnover time k to be *saturated* if $|\{j \in V | \tau_j = k\}| \geq k$. In the first instance we retain the set of vertices V_1 with saturated turnover times, and in the second all vertices V_2 with unsaturated turnover times. Now if all turnover times are saturated, then $\tau_{max} = O(n)$ and we can find a $O(\log n)$-approximation using Theorem 6. So what remains is to find a $O(\log n)$-approximation for the second instance.

Since no turnover time is saturated, it is easy to see that we can partition the vertices in V_2 into $\lceil \log n \rceil$ sets $W_1, W_2, W_4, \ldots, W_{2^{\lceil \log n \rceil}}$, such that $|W_i| \leq i$, and such that $\tau_j \geq i$ for all $j \in W_i$. For example we could first add all vertices j with $\tau_j = i$ to W_i for $i \leq \lceil \log n \rceil$, and then arbitrarily distribute vertices j with $\tau_j > \lceil \log n \rceil$ among the sets that have space. We now produce a schedule by visiting all clients in any set W_k on different days. This is feasible and implies that at most $\log n$ clients are visited on a given day, which leads to $O(\log n)$-approximation factor, as required.

The approach of Theorem 6 does not trivially extend to the MIN-AVG case. However, we may combine our result on trees with the FRT tree embeddings [13], to get a randomized $O(\log n)$-approximation.

A more direct, and deterministic $O(\log n)$-approximation is possible as well. In particular, we use the simple heuristic of visiting each client on every day that is a multiple of its turnover time, when turnover times are powers of 2. We call such a schedule a *synchronized* solution, and show that gives a logarithmic approximation.

The proof of this approximation factor, which is not trivial, works by show that a synchronized schedule is no more costly than a *non-decreasing* schedule, in which all tours are routed along a tree with turnover times non-decreasing from the root. We then show how to transform any schedule to a non-decreasing one, losing a logarithmic factor in the process. As a byproduct we show that the analysis is tight, and that a non-decreasing schedule must be $\Omega(\log n)$ times more costly than OPT in the worst case. The deterministic proof of Theorem 8 can be found in the appendix of the full paper [4].

Theorem 8. MIN-AVG RFTT *has an* $O(\log n)$*-approximation.*

Proof. We will apply the FRT tree embedding [13] of the initial instance and then use the 2-approximation for tree metrics to obtain the final solution. Given the instance (G, τ), let T be a random tree produced by the tree metric approximation with $O(\log n)$ distortion. Then $d_G(u,v) \leq d_T(u,v)$ and $E[d_T(u,v)] \leq O(\log n) d_G(u,v)$. Let S be the solution produced by the 2-approximation for MIN-AVG RFTT on the tree metric T. Then $E[S] \leq 2E[OPT(T,\tau)] \leq O(\log n)E[OPT(G,\tau)]$ by linearity of expectation on the sum over the edges.

It is an open question whether there exists a constant factor approximation algorithm for the general case. We observe that the approach of first finding a tree spanning all vertices and then using the algorithm of Sect. 3 is unsuccessful. In fact there exist instances of the problem on a graph G with n vertices, such that if we limit our attention to tours that for each day use only edges of a spanning tree of G then the obtained solution is $\Omega(\log n)$ approximated. This implies that we need some new ideas, in order to improve the $O(\log n)$ approximation of the previous theorem.

5 Conclusion

In this paper, we considered replenishment problems with fixed turnover times, a natural inventory-routing problem that has not been studied before. We formally defined the RFTT problem and considered the objectives MIN-AVG and MIN-MAX. For the MIN-AVG RFTT, we showed that it is at least as hard as the intractable PINWHEEL SCHEDULING PROBLEM on series-parallel graphs and we gave a 2-approximation for trees. For the MIN-MAX objective we showed NP-hardness on stars and gave a 6-approximation for tree metrics. We also presented a DP that solved the MIN-AVG RFTT in pseudopolynomial time on line graphs. Finally, we gave a $O(\log n)$-approximation for the MIN-MAX objective on general metrics.

The results that we present should be considered as a first step in this area and many problems remain open. An intriguing open problem is the complexity of the of RFTT on a tree. Namely, for MIN-AVG variant we conjecture that the problem is hard, and we ask whether the simple 2-approximation we provide can be improved. For the MIN-MAX variant it is open whether the problem is APX-hard and whether we can improve the 6-approximation.

Next to replenishing locations with routing aspects as we studied in this paper, scheduling problems modeling maintenance or security control of systems, form a class of problems to which this model naturally applies. It would be interesting to study such fixed turnover time problems in combination with scheduling. Would this combination allow for more definitive results?

Acknowledgments. The research of YJ and RR is supported in part by the U. S. National Science Foundation under award numbers CCF-1527032 and CCF-1655442. The research of MvE was done while he was employed by Vrije Universiteit Amsterdam.

References

1. Baruah, S., Goossens, J.: Scheduling real-time tasks: algorithms and complexity. In: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2003)
2. Baruah, S., Rosier, L., Tulchinsky, I., Varvel, D.: The complexity of periodic maintenance. In: Proceedings of the International Computer Symposium, pp. 315–320 (1990)
3. Bonifaci, V., Marchetti-Spaccamela, A.: Feasibility analysis of sporadic real-time multiprocessor task systems. Algorithmica **63**(4), 763–780 (2012)
4. Bosman, T., van Ee, M., Jiao, Y., Marchetti-Spaccamela, A., Ravi, R., Stougie, L.: Approximation algorithms for replenishment problems with fixed turnover times. arXiv preprint arXiv:1712.05218 (2017)
5. Calabro, C., Impagliazzo, R., Kabenets, V., Paturi, R.: The complexity of unique k-SAT: an isolation lemma for k-CNFs. J. Comput. Syst. Sci. **74**(3), 386–393 (2008)
6. Chan, M.Y., Chin, F.Y.L.: General schedulers for the pinwheel problem based on double-integer reduction. IEEE Trans. Comput. **41**(6), 755–768 (1992)
7. Chan, M.Y., Chin, F.Y.L.: Schedulers for larger classes of pinwheel instances. Algorithmica **9**(5), 425–462 (1993)
8. Coelho, L.C., Cordeau, J.-F., Laporte, G.: Thirty years of inventory routing. Transp. Sci. **48**(1), 1–19 (2013)
9. Coene, S., Spieksma, F.C.R., Woeginger, G.J.: Charlemagne's challenge: the periodic latency problem. Oper. Res. **59**(3), 674–683 (2011)
10. Dell, H., Husfeldt, T., Marx, D., Taslaman, N., Wahlén, M.: Exponential time complexity of the permanent and the tutte polynomial. ACM Trans. Algorithms **10**(4), 21:1–21:32 (2014)
11. Eisenbrand, F., Hähnle, N., Niemeier, M., Skutella, M., Verschae, J., Wiese, A.: Scheduling periodic tasks in a hard real-time environment. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 299–311. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14165-2_26

12. Ekberg, P., Yi, W.: Schedulability analysis of a graph-based task model for mixed-criticality systems. Real-Time Syst. **52**(1), 1–37 (2016)
13. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**(3), 485–497 (2004)
14. Fishburn, P.C., Lagarias, J.C.: Pinwheel scheduling: achievable densities. Algorithmica **34**(1), 14–38 (2002)
15. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. In: Proceedings of the 17th International Symposium on Foundations of Computer Science, pp. 216–227 (1976)
16. Gąsieniec, L., Klasing, R., Levcopoulos, C., Lingas, A., Min, J., Radzik, T.: Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) SOFSEM 2017. LNCS, vol. 10139, pp. 229–240. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51963-0_18
17. Holte, R., Mok, A., Rosier, L., Tulchinsky, I., Varvel, D.: The pinwheel: a real-time scheduling problem. In: Proceedings of the 22th Annual Hawaii International Conference on System Sciences, vol. 2, pp. 693–702 (1989)
18. Jacobs, T., Longo, S.: A new perspective on the windows scheduling problem. arXiv preprint arXiv:1410.7237 (2014)
19. Mok, A., Rosier, L., Tulchinksy, I., Varvel, D.: Algorithms and complexity of the periodic maintenance problem. Microprocess. Microprogr. **27**(1–5), 657–664 (1989)

Maximum Box Problem on Stochastic Points

Luis Evaristo Caraballo[1], Pablo Pérez-Lantero[2(✉)], Carlos Seara[3], and Inmaculada Ventura[1]

[1] Dept. de Matemática Aplicada II, Universidad de Sevilla, Seville, Spain
{lcaraballo,iventura}@us.es

[2] Dept. de Matemática y Ciencia de la Computación, Universidad de Santiago, Santiago, Chile
pablo.perez.l@usach.cl

[3] Dept. de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain
carlos.seara@upc.edu

Abstract. Given a finite set of weighted points in $\mathbb{R}^d$ (where there can be negative weights), the maximum box problem asks for an axis-aligned rectangle (i.e., box) such that the sum of the weights of the points that it contains is maximized. We consider that each point of the input has a probability of being present in the final random point set, and these events are mutually independent; then, the total weight of a maximum box is a random variable. We aim to compute both the probability that this variable is at least a given parameter, and its expectation. We show that even in $d = 1$ these computations are #P-hard, and give pseudo polynomial-time algorithms in the case where the weights are integers in a bounded interval. For $d = 2$, we consider that each point is colored red or blue, where red points have weight $+1$ and blue points weight $-\infty$. The random variable is the maximum number of red points that can be covered with a box not containing any blue point. We prove that the above two computations are also #P-hard, and give a polynomial-time algorithm for computing the probability that there is a box containing exactly two red points, no blue point, and a given point of the plane.

1 Introduction

Let $P \subset \mathbb{R}^d$ be a finite point set of n points, where each point is assigned a positive or negative weight. The *maximum box* problem receives P and outputs an axis-aligned hyperrectangle (i.e., box) such that the sum of the weights of the points of P that it contains is maximized; and it can be solved in $O(n^d)$ time [4].

This work is the union of two works that appear in the book of abstracts of the XVII Spanish Meeting on Computational Geometry, Alicante, Spain, 2017. The extended version is available at https://sites.google.com/a/usach.cl/pablo/maxbox.pdf.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 734922.

We consider the maximum box problem on a recent uncertainty model in which each input point has assigned a probability of being present in the final (hence random) point set. Particularly, each point $p \in P$ has assigned the probability $\pi(p) \in [0,1]$ and we consider the random point set $S \subseteq P$ where each point $p \in P$ is included in S independently and uniformly at random with probability $\pi(p)$. Let $\mathsf{box}(S)$ denote the total weight of the points of S covered by a maximum box of S, which is now a random variable. Then, one can ask the following questions: What is the probability that for the final point set there exists a box that covers a weight sum at least a given parameter k (i.e., compute $\Pr[\mathsf{box}(S) \geq k]$)? What is the expectation of the maximum weight sum that can be covered with a box (i.e., compute $\mathbb{E}[\mathsf{box}(S)]$)?

Uncertainty models come from real scenarios in which the big amount of data, arriving from many sources, have inherent uncertainty. In computational geometry, we can find several recent works on uncertain point sets such as: the expected total length of the minimum Euclidean spanning tree [6]; the probability that the distance between the closest pair of points is at least a given parameter [11]; the computation of the most-likely convex hull [13]; the probability that the area or perimeter of the convex hull is at least a given parameter [12]; the center minimizing the maximum expected distance from the points [9]; the probability that the 2-colored point set is linearly separable [10]; and data structures for range-max queries on uncertain data [1].

Note that

$$\Pr[\mathsf{box}(S) \geq k] = \sum_{X \subseteq P} \mathbf{1}_{[\mathsf{box}(X) \geq k]}(X) \cdot \Pr[S = X]$$

and

$$\mathbb{E}[\mathsf{box}(S)] = \sum_{X \subseteq P} \mathsf{box}(X) \cdot \Pr[S = X],$$

where

$$\Pr[S = X] = \prod_{p \in X} \pi(p) \cdot \prod_{p \in P \setminus X} (1 - \pi(p)).$$

Hence, since for any $X \subseteq P$ we can compute $\mathsf{box}(X)$ in $O(|X|^d) = O(n^d)$ time for fixed d, both $\Pr[\mathsf{box}(S) \geq k]$ and $\mathbb{E}[\mathsf{box}(S)]$ can be trivially computed exactly in exponential time. We show that even in $d = 1$ the exact computations of these values are #P-hard problems. To estimate them with high probability of success, we can use standard Monte-Carlo methods in which we generate a polynomial number of outcomes $X \subseteq P$ and compute $\mathsf{box}(X)$ for each of them.

For $d = 1$, the maximum box problem asks for an interval of the line. If the points are uncertain as described above, then it is equivalent to consider as input a sequence of random numbers, where each number has two possible outcomes: zero if the number is not present and the actual value of the number otherwise. The output is the subsequence of consecutive numbers with maximum sum. We consider the simpler case when the subsequence is a partial sum, that is, it contains the first (or leftmost) number of the sequence. More formally: We say that a random variable X is *zero-value* if $X = v$ with probability ρ, and $X = 0$ with probability $1 - \rho$, for an integer number $v = v(X) \neq 0$ and a probability ρ. We refer to v as the value of X and to ρ as the probability of X. In any sequence of zero-value variables, all variables are assumed to be mutually independent. Let $\mathcal{X} = X_1, X_2, \ldots, X_n$ be a sequence of n mutually independent zero-value variables, whose values are $a_1, a_2, \ldots, a_n$, respectively. We study the random variable $\mathsf{S}(\mathcal{X}) = \max\{0, X_1, X_1 + X_2, \ldots, X_1 + \cdots + X_n\}$, which is the maximum partial sum of the random sequence $\mathcal{X}$. We prove (Sect. 2.1) that computing $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ for any fixed $z \geq 1$, and computing the expectation $\mathbb{E}[\mathsf{S}(\mathcal{X})]$ are both #P-hard problems, even if all variables of $\mathcal{X}$ have the same positive, less-than-one probability. When $a_1, a_2, \ldots, a_n \in [-a..b]$ for bounded $a, b \in \mathbb{N}$, we show (Sect. 2.2) that both $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ and $\mathbb{E}[\mathsf{S}(\mathcal{X})]$ can be computed in time polynomial in n, a, and b.

For $d = 2$, we consider the maximum box problem in the context of red and blue points, where red points have weight $+1$ and blue points weight $-\infty$. Let R and B be disjoint finite point sets in the plane with a total of n points, where the elements of R are colored red and the elements of B are colored blue. The maximum box problem asks for a box H such that $|H \cap R|$ is maximized subject to $|H \cap B| = \emptyset$. This problem has been well studied, with algorithms whose running times go from $O(n^2 \log n)$ [7], $O(n^2)$ [4], to $O(n \log^3 n)$ [3]. In our uncertainty model, $\mathsf{box}(S)$ denotes the random variable equal to the maximum number of red points in the random point set $S \subseteq R \cup B$ that can be covered with a box not covering any blue point of S. We prove (Sect. 3.1) that computing the probability $\Pr[\mathsf{box}(S) \geq k]$ for any given $k \geq 2$, and computing the expectation $\mathbb{E}[\mathsf{box}(S)]$, are both #P-hard problems. We further show (Sect. 3.2) that given a point o of the plane, computing the probability that there exists a box containing exactly two red points of S as opposite vertices, no blue point of S, and the point o can be solved in polynomial time. If we remove the restriction of containing o, this problem is also #P-hard.

2 Weighted Points in One Dimension

2.1 Hardness

Theorem 1. *For any integer $z \geq 1$ and any $\rho \in (0, 1)$, given a sequence $\mathcal{X} = X_1, X_2, \ldots, X_n$ of n zero-value random variables, each with probability ρ, it is #P-hard to compute* $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$.

Proof. Let $z \geq 1$ be an integer, and $\rho \in (0, 1)$ a probability. We show a Turing reduction from the **#SubsetSum** problem, which is known to be #P-complete [8]. Our reduction assumes an unknown algorithm (i.e., oracle) $\mathcal{A}(\mathcal{X})$ computing $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ for any finite sequence $\mathcal{X}$ of zero-value random variables, that will be called twice. **#SubsetSum** receives as input a set $\{a_1, \ldots, a_n\} \subset \mathbb{N}$ of n numbers and a target $t \in \mathbb{N}$, and counts the number of subsets $J \subseteq [1..n]$ such that $\sum_{j \in J} a_j = t$. It remains #P-hard if the subsets J

to count must also satisfy $|J| = k$, for given $k \in [1..n]$. Let $(\{a_1, \ldots, a_n\}, t, k)$ be an instance of this **#SubsetSum**, in which we assume $t \leq a_1 + \cdots + a_n$.

Let $m = \max\{z, 1 + a_1 + \cdots + a_n\} > t$, and $\mathcal{X} = X_0, X_1, X_2, \ldots, X_n$ be a sequence of $n+1$ zero-value random variables, each with probability ρ, where the value of X_0 is $-km - t + z$, and the value of X_i is $m + a_i$ for every $i \in [1..n]$. Observe that for $J \subseteq [1..n]$ we have

$$\sum_{j \in J}(m + a_j) = km + t \;\Leftrightarrow\; \left(\sum_{j \in J} a_j = t \text{ and } |J| = k\right).$$

Furthermore, $|J| > k$ implies $\sum_{j \in J}(m + a_j) > km + t$. Let $J_{\mathcal{X}} = \{j \in [1..n] : X_j \neq 0\}$, and for any s, let $N_s = |J \subseteq [1..n] : |J| = k, \sum_{j \in J} a_j \geq s|$. Then, the **#SubsetSum** asks for $N_t - N_{t+1}$. Call $\mathcal{A}(\mathcal{X})$ to compute $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$. Then:

$$\Pr[\mathsf{S}(\mathcal{X}) \geq z] = \Pr[\mathsf{S}(\mathcal{X}) \geq z, X_0 = 0] + \Pr[\mathsf{S}(\mathcal{X}) \geq z, X_0 = -km - t + z]$$

where,

$$\begin{aligned}\Pr[\mathsf{S}(\mathcal{X}) \geq z, X_0 = 0] &= \Pr[X_0 = 0] \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq z \mid X_0 = 0] \\ &= (1-\rho) \cdot \Pr[|J_{\mathcal{X}}| \geq 1] = (1-\rho) \cdot (1 - \Pr[|J_{\mathcal{X}}| = 0]) \\ &= (1-\rho) \cdot (1 - (1-\rho)^n),\end{aligned}$$

and

$$\begin{aligned}&\Pr[\mathsf{S}(\mathcal{X}) \geq z, X_0 = -km - t + z] \\ &= \Pr[X_0 = -km - t + z] \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq z \mid X_0 = -km - t + z] \\ &= \rho \cdot \Pr\left[-km - t + z + \sum_{j \in J_{\mathcal{X}}}(m + a_j) \geq z\right] = \rho \cdot \Pr\left[\sum_{j \in J_{\mathcal{X}}}(m + a_j) \geq km + t\right] \\ &= \rho \cdot \left(\Pr\left[|J_{\mathcal{X}}| = k, \sum_{j \in J_{\mathcal{X}}}(m + a_j) \geq km + t\right]\right. \\ &\left.+ \sum_{i=k+1}^{n} \Pr\left[|J_{\mathcal{X}}| = i, \sum_{j \in J_{\mathcal{X}}}(m + a_j) \geq km + t\right]\right) \\ &= \rho \cdot \Pr\left[|J_{\mathcal{X}}| = k, \sum_{j \in J_{\mathcal{X}}} a_j \geq t\right] + \rho \cdot \sum_{i=k+1}^{n} \Pr[|J_{\mathcal{X}}| = i] \\ &= \rho \cdot N_t \cdot \rho^k \cdot (1-\rho)^{n-k} + \rho \cdot \sum_{i=k+1}^{n} \binom{n}{i} \cdot \rho^i \cdot (1-\rho)^{n-i}.\end{aligned}$$

Hence, we can compute N_t in polynomial time from the value of $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$. Consider now the random sequence $\mathcal{X}' = X'_0, X_1, X_2, \ldots, X_n$, where X'_0 has value $-km - (t+1) + z$. Using arguments similar as above, by calling $\mathcal{A}(\mathcal{X}')$

to compute $\Pr[\mathsf{S}(\mathcal{X}') \geq z]$, we can compute N_{t+1} in polynomial time from this probability. Then, $N_t - N_{t+1}$ can be computed in polynomial time, plus the time of calling twice the oracle $\mathcal{A}$. This implies the theorem. □

Theorem 2. *For any $\rho \in (0,1)$, given a sequence $\mathcal{X} = X_1, \dots, X_n$ of n zero-value random variables, each with probability ρ, it is #P-hard to compute $\mathbb{E}[\mathsf{S}(\mathcal{X})]$.*

Proof. Let $\mathcal{X} = X_1, X_2, \dots, X_n$ be any sequence of zero-value random variables, each with probability ρ, and consider the sequence $\mathcal{X}' = X_0, X_1, \dots, X_n$, where X_0 is a zero-value random variable with value -1 and probability ρ. Let w be the sum of the positive values among the values of $X_1, \dots, X_n$. Then:

$$\mathbb{E}[\mathsf{S}(\mathcal{X})] = \sum_{i=1}^{w} i \cdot \Pr[\mathsf{S}(\mathcal{X}) = i] = \sum_{i=1}^{w} \Pr[\mathsf{S}(\mathcal{X}) \geq i],$$

and

$$\begin{aligned}
\mathbb{E}[\mathsf{S}(\mathcal{X}')] &= \sum_{i=1}^{w} \Pr[\mathsf{S}(\mathcal{X}') \geq i] \\
&= \sum_{i=1}^{w} (\Pr[\mathsf{S}(\mathcal{X}') \geq i, X_0 = 0] + \Pr[\mathsf{S}(\mathcal{X}') \geq i, X_0 = -1]) \\
&= \sum_{i=1}^{w} (\Pr[X_0 = 0] \cdot \Pr[\mathsf{S}(\mathcal{X}') \geq i \mid X_0 = 0] \\
&\quad + \Pr[X_0 = -1] \cdot \Pr[\mathsf{S}(\mathcal{X}') \geq i \mid X_0 = -1]) \\
&= \sum_{i=1}^{w} ((1-\rho) \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq i] + \rho \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq i+1]) \\
&= \sum_{i=1}^{w} (1-\rho) \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq i] + \sum_{i=2}^{w+1} \rho \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq i] \\
&= (1-\rho) \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq 1] + \sum_{i=2}^{w} \Pr[\mathsf{S}(\mathcal{X}) \geq i].
\end{aligned}$$

Then, we have that $\mathbb{E}[\mathsf{S}(\mathcal{X})] - \mathbb{E}[\mathsf{S}(\mathcal{X}')] = \rho \cdot \Pr[\mathsf{S}(\mathcal{X}) \geq 1]$. Since computing $\Pr[\mathsf{S}(\mathcal{X}) \geq 1]$ is #P-hard (Theorem 1), then computing $\mathbb{E}[\mathsf{S}(\mathcal{X})]$ is also #P-hard via a Turing reduction. □

2.2 Pseudo-Polynomial Time Algorithms

Let $\mathcal{X} = X_1, X_2, \dots, X_n$ be a sequence of n random zero-value variables, with values $a_1, a_2, \dots, a_n \in [-a..b] \subset \mathbb{Z}$ and probabilities $\rho_1, \rho_2, \dots, \rho_n$, respectively, for some $a, b \in \mathbb{N}$. We show that both $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ and $\mathbb{E}[\mathsf{S}(\mathcal{X})]$ can be computed in time polynomial in n, a, and b. Let $J = \{j \in [1..n] : a_j < 0\}$ and

$$w_0 = \sum_{j \in J} |a_j| = O(na) \text{ and } w_1 = \sum_{j \in [1..n] \setminus J} a_j = O(nb).$$

For every $t \in [1..n]$, let

$$S_t = X_1 + \cdots + X_t, \; M_t = \max\{0, S_1, S_2, \ldots, S_t\}, \text{and}$$
$$G_t = \{\Pr[M_t = k, S_t = s] : k \in [0..w_1], s \in [-w_0..w_1], k \geq s\}.$$

Observe that G_t has size $O(w_1(w_0 + w_1)) = O(nb(na + nb)) = O(n^2b(a + b))$ for every t, and that G_1 can be trivially computed. Using the dynamic programming algorithm design paradigm, we next show how to compute the values of G_t, $t \geq 2$, assuming that all values of G_{t-1} have been computed. Note that:

$$\Pr[M_t = k, S_t = s] = \Pr[M_t = k, S_t = s, X_t = 0] + \Pr[M_t = k, S_t = s, X_t = a_t],$$

where

$$\begin{aligned}\Pr[M_t = k, S_t = s, X_t = 0] &= \Pr[X_t = 0] \cdot \Pr[M_t = k, S_t = s \mid X_t = 0] \\ &= (1 - \rho_t) \cdot \Pr[M_{t-1} = k, S_{t-1} = s]\end{aligned}$$

and

$$\begin{aligned}\Pr[M_t = k, S_t = s, X_t = a_t] &= \Pr[X_t = a_t] \cdot \Pr[M_t = k, S_t = s \mid X_t = a_t] \\ &= \rho_t \cdot \Pr[M_t = k, S_t = s \mid X_t = a_t].\end{aligned}$$

When $k = s$, we have for $a_t < 0$ that $\Pr[M_t = k, S_t = s \mid X_t = a_t] = 0$, since this event indicates that $S_t = X_1 + \cdots + X_t$ is a maximum partial sum of $X_1, \ldots, X_t$, but this cannot happen because any maximum partial sum ends in a positive element. For $a_t > 0$ we have

$$\begin{aligned}\Pr[M_t = k, S_t = s \mid X_t = a_t] &= \Pr[M_{t-1} \leq k, S_{t-1} = s - a_t] \\ &= \sum_{i=s-a_t}^{k} \Pr[M_{t-1} = i, S_{t-1} = s - a_t].\end{aligned}$$

When $k > s$, M_t does not count the element a_t, hence $M_{t-1} = M_t$. Then

$$\Pr[M_t = k, S_t = s \mid X_t = a_t] = \Pr[M_{t-1} = k, S_{t-1} = s - a_t].$$

Modeling each set G_t as a 2-dimensional table (or array), note that each value of G_t can be computed in $O(k - (s - a_t)) = O(w_1)$ time, and hence all values of G_t can be computed in $O(w_1) \cdot O(n^2b(a+b)) = O(n^3b^2(a+b))$ time. Finally, once all the values of G_n have been computed in $O(n) \cdot O(n^3b^2(a + b)) = O(n^4b^2(a + b))$ time, we can compute $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ as

$$\Pr[\mathsf{S}(\mathcal{X}) \geq z] = \sum_{k=z}^{w_1} \Pr[\mathsf{S}(\mathcal{X}) = k] = \sum_{k=z}^{w_1} \sum_{s=-w_0}^{k} \Pr[M_n = k, S_n = s]$$

in $O(w_1(w_0 + w_1)) = O(n^2b(a + b))$ time, and $\mathbb{E}[\mathsf{S}(\mathcal{X})] = \sum_{z=1}^{w_1} \Pr[\mathsf{S}(\mathcal{X}) \geq z]$ in $O(w_1) = O(nb)$ time. As a consequence, we get the following result.

Theorem 3. *Let $\mathcal{X}$ be a sequence of n random zero-value variables, with values in the range $[-a..b] \subset \mathbb{Z}$ for some $a, b \in \mathbb{N}$. Then, both $\Pr[\mathsf{S}(\mathcal{X}) \geq z]$ and $\mathbb{E}[\mathsf{S}(\mathcal{X})]$ can be computed in time polynomial in n, a, and b.*

3 Red and Blue Points in the Plane

In this section, we show that computing $\Pr[\mathsf{box}(S) \geq k]$ and $\mathbb{E}[\mathsf{box}(S)]$ when $S \subseteq R \cup B$ is taken at random, are both #P-hard. To do that, we will show a one-to-many reduction from the problem of counting the number of independent sets in a planar bipartite graph with maximum degree 4. Given such a graph G, we will generate a polynomial number of inputs (i.e., random colored point sets) for the problem of computing $\Pr[\mathsf{box}(S) \geq k]$ (or $\mathbb{E}[\mathsf{box}(S)]$), where each input is associated with a different graph G_s for some $s \geq 1$, obtained by adding s vertices to each edge of G. For every input, the number $N(G_s)$ of independent sets of G_s can be computed in polynomial time, plus a call to an oracle computing $\Pr[\mathsf{box}(S) \geq k]$ (or $\mathbb{E}[\mathsf{box}(S)]$). Before the reduction, it is shown that $N(G)$ (the number of independent sets of G) can be computed in polynomial time from the values of $N(G_s)$ for all s. We complement these hardness results with a polynomial-time algorithm to compute the probability that there exists a box restricted to contain a given point $o \notin R \cup B$ of the plane, two red points as opposite vertices, and no blue point.

3.1 Hardness

Given a graph $G = (V, E)$, a subset $V' \subseteq V$ is an *independent set* of G if no pair of vertices of V' define en edge in E. Let $N(G)$ denote the number of independent sets of G. The problem **#IndSet** of counting the number of independent sets in a graph is #P-complete, even if the graph is planar, bipartite, and with maximum degree 4 [14]. We show a one-to-many Turing reduction from **#IndSet** to the problem of computing $\Pr[\mathsf{box}(S) \geq k]$, for any given $k \geq 2$. Let $G = (V, E)$ be the input of **#IndSet**, where G is a planar bipartite graph with maximum degree 4. Let $n = |V|$ and $m = |E| = O(n)$.

For any subset $V' \subseteq V$ and any edge $e = \{u, v\} \in E$, we say that V' *1-covers* edge e if exactly one of u and v belongs to V'. We also say that V' *2-covers* e if both u and v belong to V'. Let $C_{i,j}$ denote the number of subsets of V that 1-cover exactly i edges and 2-cover exactly j edges. Then, $N(G) = \sum_{i=0}^{m} C_{i,0}$.

For $s \geq 1$, let $G_s = (V_s, E_s)$ be the graph obtained from G by adding exactly s intermediate vertices on each edge of E. Let $\{f_i\}_{i=1}^{\infty}$ be the Fibonacci sequence, with $f_1 = f_2 = 1$ and $f_i = f_{i-1} + f_{i-2}$ for $i \geq 3$. Let $\alpha_i = f_{i+1}/f_{i+2}$ for $i \geq 0$.

Lemma 1. *We have*

$$N(G_s) = (f_{s+2})^m \cdot \sum_{0 \leq i+j \leq m} C_{i,j} \cdot (\alpha_s)^i \cdot (1 - \alpha_s)^j.$$

Proof. Any independent set $V'_s \subseteq V_s$ of G_s induces the subset $V'_s \cap V$ of V, which is not necessarily an independent set of G because it may 2-cover some edges. Let $V' \subseteq V$ be any subset of V that 1-covers i edges and 2-covers j edges. For any edge $e \in E$, let p_e denote the path of G_s induced by the s vertices added to e when constructing G_s from G. An independent set of G_s inducing V' can be obtained by starting with V' and adding vertices in the following way. For every edge $e = \{u, v\} \in E$:

(1) if V' neither 1-covers nor 2-covers e, then add any independent set of p_e.
(2) if V' 1-covers e, say $u \in V'$, then add any independent set of p_e not containing the extreme vertex of p_e adjacent to u in G_s.
(3) if V' 2-covers e, then add any independent set of p_e with no extreme vertex.

It is well known that the number of independent sets of a path of length ℓ is exactly $f_{\ell+3}$ [14]. For example, if $\ell = 1$ then the path is an edge $\{u, v\}$, and has $f_{1+3} = f_4 = 3$ independent sets: $\{\}$, $\{u\}$, and $\{v\}$. Since p_e has length $s - 1$ for every e, the number of choices for cases (1), (2), and (3) are f_{s+2}, f_{s+1}, and f_s, respectively. Therefore, the number of independent sets of G_s inducing a subset of V that 1-covers i edges and 2-covers j edges is precisely $C_{i,j} \cdot (f_{s+1})^i \cdot (f_s)^j \cdot (f_{s+2})^{m-i-j}$. Hence, the number $N(G_s)$ of independent sets of G_s satisfies

$$\begin{aligned}
N(G_s) &= \sum_{0 \le i+j \le m} C_{i,j} \cdot (f_{s+1})^i \cdot (f_s)^j \cdot (f_{s+2})^{m-i-j} \\
&= (f_{s+2})^m \cdot \sum_{0 \le i+j \le m} C_{i,j} \cdot \left(\frac{f_{s+1}}{f_{s+2}}\right)^i \cdot \left(\frac{f_s}{f_{s+2}}\right)^j \\
&= (f_{s+2})^m \cdot \sum_{0 \le i+j \le m} C_{i,j} \cdot \left(\frac{f_{s+1}}{f_{s+2}}\right)^i \cdot \left(1 - \frac{f_{s+1}}{f_{s+2}}\right)^j \\
&= (f_{s+2})^m \cdot \sum_{0 \le i+j \le m} C_{i,j} \cdot (\alpha_s)^i \cdot (1 - \alpha_s)^j,
\end{aligned}$$

which completes the proof. □

Lemma 2. *Let T be a set of $m + 1$ integers, each in the range $[1..n^c]$ for some constant $c > 0$. If we know the value of $N(G_s)$ for every $s \in T$, then the number $N(G)$ can be computed in time polynomial in n.*

Proof. For every $s \in T$, the value of $(f_{s+2})^m$ can be computed in $O(\log s + \log m) = O(\log n)$ time, and the value of α_s also in $O(\log s) = O(\log n)$ time. Let $b_s = N(G_s)/(f_{s+2})^m$ for every $s \in T$. Consider the polynomial

$$P(x) = \sum_{0 \le i+j \le m} C_{i,j} \cdot x^i \cdot (1 - x)^j = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m,$$

of degree m, whose coefficients $a_0, a_1, \ldots, a_m$ are linear combinations of the terms $C_{i,j}$. By Lemma 1, and using the known values of b_s and α_s for every $s \in T$, we have $m + 1$ evaluations of $P(x)$ of the form $b_s = P(\alpha_s)$, each corresponding to the linear equation $b_s = a_0 + a_1 \cdot \alpha_s + a_2 \cdot \alpha_s^2 + \cdots + a_m \cdot \alpha_s^m$ with variables the coefficients $a_0, a_1, \ldots, a_m$. The main matrix of this system of $m + 1$ linear equations is Vandermonde, with parameters α_s for every $s \in T$. All α_s are distinct [14], then the determinant of the main matrix is non-zero, and the system has a unique solution $a_0, a_1, \ldots, a_m$ which can be computed in time polynomial in n. Finally,

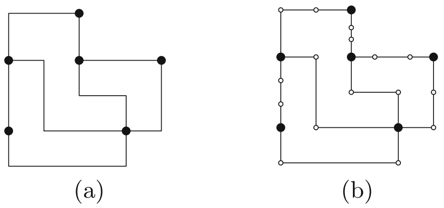


Fig. 1. (a) An embedding of G. (b) The embedding of G_s for $s = 2$: two intermediate vertices are added to each edge of G so that all polyline bends are covered.

observe that for $j = 0$, the coefficient of the polynomial $C_{i,j} \cdot x^i \cdot (1-x)^j = C_{i,0} \cdot x^i$ is $C_{i,0}$. Furthermore, for $j > 0$, all the coefficients of the polynomial

$$C_{i,j} \cdot x^i \cdot (1-x)^j = C_{i,j} \cdot x^i \cdot \left(\binom{j}{0} - \binom{j}{1} x^1 + \binom{j}{2} x^2 - \cdots + (-1)^j \binom{j}{j} x^j \right)$$

sum up to zero. Hence, $a_0 + a_1 + \cdots + a_m = \sum_{i=0}^{m} C_{i,0} = N(G)$ which shows that $N(G)$ can be computed in time polynomial in n. $\square$

In polynomial time, the graph $G = (V, E)$ can be embedded in the plane using $O(n^2)$ area in such a way that its vertices are at integer coordinates, and its edges are drawn so that they are polylines made up of line segments of the form $x = i$ or $y = j$, for integers i and j [15] (see Fig. 1a). Let $s_0 = O(n)$ be the maximum number of bends of the polylines corresponding to the edges.

For an arbitrary $s \in \mathbb{N}$, such that $s \geq s_0$ and $s = O(n)$, we embed the graph G_s in the following way. We embed the graph G as above; scale the embedding by factor $2(s+1)$; and for each edge of G, add s intermediate vertices to the polyline of the edge so that they have even integer coordinates and cover all bends of the polyline (see Fig. 1b). Then, each edge of G_s is represented in the embedding by a vertical or horizontal segment. Let the point set $R_0 = R_0(s) \subset \mathbb{Z}^2$ denote the vertex set of the embedding, and color these points in red. By translation if necessary, we can assume $R_0 \subset [0..N]^2$ for some $N = O(n^2)$. Let $B_0 = B_0(s)$ be the next set of blue points: For each horizontal or vertical line ℓ through a point of R_0, and each two consecutive points $p, q \in R_0$ in ℓ such that the vertices p and q are not adjacent in G_s, we add a blue point in the segment pq connecting p and q so that it has one odd coordinate. Note that $|B_0| = O(|R_0|) = O(n + m \cdot s) = O(n^2)$. Now, a horizontal or vertical segment connecting two points p and q of $R_0 \cup B_0$ represents an edge of G_s *if and only if* $p, q \in R_0$ and the segment does not contain any other point of $R_0 \cup B_0$ in its interior.

We would like that two red points of R_0 can be covered with a box avoiding blue points if and only if the two red points represent an edge of G_s. To achieve this, we perturb the elements of $R_0 \cup B_0$ and add extra blue points.

We perturb $R_0 \cup B_0 \subset [0..N]^2$ to obtain a point set in general position (with rational coordinates) by applying the function $\lambda : [0..N]^2 \to \mathbb{Q}^2$, where

$$\lambda(p) = \left(x(p) + \frac{x(p) + y(p)}{4N + 1}, y(p) + \frac{x(p) + y(p)}{4N + 1} \right),$$

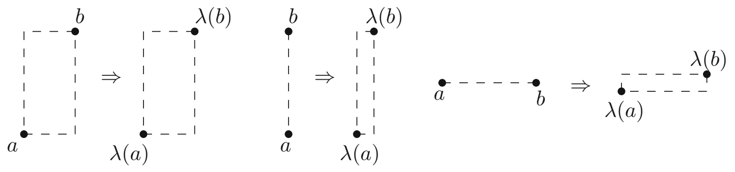


Fig. 2. The way in which points are perturbed using function λ.

to every $p \in R_0 \cup B_0$, where $x(p)$ and $y(p)$ denote the x- and y-coordinates of p, respectively. Similar perturbations can be found in [2,5], and refer to Fig. 2. Since λ is injective [5], let λ^{-1} denote the inverse of λ. For $X \subset [0..N]^2$, let $\lambda(X) = \{\lambda(p) \mid p \in X\}$, and for $Y \subset \lambda([0..N]^2)$ let $\lambda^{-1}(Y) = \{\lambda^{-1}(p) \mid p \in Y\}$. Let $\delta = 1/(4N+2)$, and define the sets

$$R = \lambda(R_0) \text{ and } B = \lambda(B_0) \cup \{p + (1/2, 1/2), p + (\delta, -\delta) \mid p \in R\}\,.$$

Note that $|R| = O(n^2)$ and $|B| = O(n^2)$. For two points a and b, let $D(a,b)$ be the box with the segment ab as a diagonal.

Lemma 3. *For any different $p, q \in R$, the box $D(p,q)$ contains no points of B if and only if the vertices $\lambda^{-1}(p)$ and $\lambda^{-1}(q)$ are adjacent in G_s.*

The proof of Lemma 3 is deferred to the extended version.

Theorem 4. *Given $R \cup B$, it is #P-hard to compute* $\Pr[\mathsf{box}(S) \geq k]$ *for every integer $k \geq 2$, and it is also #P-hard to compute* $\mathbb{E}[\mathsf{box}(S)]$.

Proof. Let $k = 2$. Assume that there exists an algorithm (i.e., oracle) $\mathcal{A}$ that computes $\Pr[\mathsf{box}(S) \geq 2]$. Consider the planar bipartite graph $G = (V, E)$, with maximum degree 4, the input of **#IndSet**. Let $T = \{s_0, s_0 + 1, \ldots, s_0 + m\}$. For each $s \in T$ we create the graph G_s, embed G_s in the plane, and create the colored point set $R \cup B$ from this embedding. To each red point $p \in R$ we set its probability $\pi(p)$ to $1/2$, and for each blue point $q \in B$ we set $\pi(q) = 1$. Note from Lemma 3 that there does not exist any box containing more than two red points of R and no blue point from B. Then, we have $\Pr[\mathsf{box}(S) \geq 2] = \Pr[\mathsf{box}(S) = 2]$, where $S \subseteq R \cup B$ is the random subset of $R \cup B$. Furthermore,

$$\begin{aligned}
\Pr[\mathsf{box}(S) = 2] &= \Pr[\lambda^{-1}(S \cap R) \text{ is not an independent set in } G_s] \\
&= 1 - \Pr[\lambda^{-1}(S \cap R) \text{ is an independent set in } G_s] \\
&= 1 - \frac{N(G_s)}{2^{|R|}} \\
N(G_s) &= 2^{|R|} \cdot (1 - \Pr[\mathsf{box}(S) \geq 2]).
\end{aligned}$$

Then, for each $s \in T$ we can compute $N(G_s)$ by calling $\mathcal{A}$ once. By Lemma 2, we can compute $N(G)$ from the $m+1$ computed values of $N(G_s)$ for each $s \in T$. Hence, it is #P-hard to compute $\Pr[\mathsf{box}(S) \geq 2]$ via a Turing reduction from

#IndSet. To show that computing $\mathbb{E}[\mathsf{box}(S)]$ is also #P-hard, for each $s \in T$ consider the above point set $R \cup B$ and note that

$$\begin{aligned}\mathbb{E}[\mathsf{box}(S)] &= 1 \cdot \Pr[\lambda^{-1}(S \cap R) \text{ is an independent set in } G_s, S \cap R \neq \emptyset] + \\ &\quad 2 \cdot \Pr[\lambda^{-1}(S \cap R) \text{ is not an independent set in } G_s] \\ &= \frac{N(G_s) - 1}{2^{|R|}} + 2 \cdot \left(1 - \frac{N(G_s)}{2^{|R|}}\right) \\ &= 2 - \frac{N(G_s) + 1}{2^{|R|}} \\ N(G_s) &= 2^{|R|} \cdot (2 - \mathbb{E}[\mathsf{box}(S)]) - 1.\end{aligned}$$

Let now $k \geq 3$. For each $s \in T$, the graph G_s can be colored with two colors, 0 and 1, because it is also a bipartite graph. Each red point in R corresponds to a vertex in G_s. Then, for each red point $p \in R$ with color 0 we add new $\lfloor \frac{k}{2} \rfloor - 1$ red points close enough to p (say, at distance much smaller than δ), and for each red point $q \in R$ with color 1 we add new $\lceil \frac{k}{2} \rceil - 1$ red points close enough to q. Let $R' = R'(s)$ be the set of all new red points, and assign $\pi(u) = 1$ for every $u \in R'$. In this new colored point set $R \cup R' \cup B$, there is no box containing more than k red points and no blue point. Furthermore, every box containing exactly k red points and no blue point contains two points $p, q \in R$ such that $\lambda^{-1}(p)$ and $\lambda^{-1}(q)$ are adjacent in G_s; and for every $p, q \in R$ such that $\lambda^{-1}(p)$ and $\lambda^{-1}(q)$ are adjacent in G_s such a box containing p and q exists. Then, when taking $S \subseteq R \cup R' \cup B$ at random, we also have

$$\begin{aligned}\Pr[\mathsf{box}(S) \geq k] &= \Pr[\mathsf{box}(S) = k] \\ &= \Pr[\lambda^{-1}(S \cap R) \text{ is not an independent set in } G_s] \\ &= 1 - N(G_s)/2^{|R|}.\end{aligned}$$

Hence, computing $\Pr[\mathsf{box}(S) \geq k]$ is also #P-hard for any $k \geq 3$. □

3.2 Two-Point Boxes

From the proof of Theorem 4, note that it is also #P-hard to compute the probability that in $S \subseteq R \cap B$ there exists a box that contains exactly 2 red points as opposite vertices and no blue point. In this section, we present a polynomial-time algorithm to compute the probability that there exists a box restricted to contain a given point $o \notin R \cup B$ of the plane, two red points as opposite vertices, and no blue point. We assume general position, that is, there are no two points of $R \cup B \cup \{o\}$ with the same x- or y-coordinate. We further assume w.l.o.g. that o is the origin of coordinates.

Given a fixed $X \subseteq R \cup B$, and $S \subseteq R \cup B$ taken at random, let $E(X) = E(X, S)$ be the event that there exists a box containing the origin o, exactly two red points in $S \cap X$ as opposite vertices, and no blue in $S \cap X$. Then, our goal is to compute $\Pr[E(R \cup B)]$.

Theorem 5. *Given $R \cup B$, $\Pr[E(R \cup B)]$ can be computed in polynomial time.*

Proof. Let $X \subseteq R \cup B$, and define $X^+ = \{p \in X \mid y(p) > 0\}$ and $X^- = \{p \in X \mid y(p) < 0\}$. Given points $q \in X^+$ and $r \in X^-$, define the events

$$U_q(X) = U_q(X,S) = \left[q = \arg \min_{p \in X^+ \cap S} \{y(p)\} \right],$$

and

$$D_r(X) = D_r(X,S) = \left[r = \arg \max_{p \in X^- \cap S} \{y(p)\} \right].$$

Let $U_q(X) = U_q(X,S)$ and $D_r(X) = D_r(X,S)$. Using the formula of the total probability, we have:

$$\begin{aligned} \Pr[E(X)] &= \sum_{q \in X^+} \Pr\left[E(X) \mid U_q(X)\right] \cdot \Pr\left[U_q(X)\right] \\ &= \sum_{q \in X^+} \Pr\left[E(X) \mid U_q(X)\right] \cdot \left(\pi(q) \cdot \prod_{p \in X^+ : y(p) < y(q)} (1 - \pi(p)) \right). \end{aligned}$$

To compute $\Pr\left[E(X) \mid U_q(X)\right]$, we assume $x(q) > 0$. The case where $x(q) < 0$ is symmetric. If $q \in B$, then observe that when restricted to the event $U_q(X)$ any box containing exactly two red points of $S \cap X$ and the origin o of coordinates, where one of these red points is to the right of q, will contain q. Hence, we must "discard" all points to the right of q, all points in between the horizontal lines through q and o because they are not present, and q itself. Hence:

$$\Pr\left[E(X) \mid U_q(X)\right] = \Pr[E(X_q)],$$

where $X_q \subset X$ contains the points $p \in X$ such that $x(p) < x(q)$ and either $y(p) > y(q)$ or $y(p) < 0$. If $q \in R$, we expand $\Pr\left[E(X) \mid U_q(X)\right]$ as follows:

$$\begin{aligned} \Pr\left[E(X) \mid U_q(X)\right] &= \sum_{r \in X^-} \Pr\left[E(X) \mid U_q(X), D_r(X)\right] \cdot \Pr\left[D_r(X)\right] \\ &= \sum_{r \in X^-} \Pr\left[E(X) \mid U_q(X), D_r(X)\right] \cdot \left(\pi(r) \cdot \prod_{p \in X^- : y(p) > y(r)} (1 - \pi(p)) \right). \end{aligned}$$

There are now three cases according to the relative positions of q and r.

Case 1: $x(r) < 0 < x(q)$. Let $Y_{q,r} \subset X$ contain the points $p \in X$ (including q) such that $x(r) < x(p) \leq x(q)$ and either $y(p) < y(r)$ or $y(p) \geq y(q)$. If $r \in R$, then $\Pr\left[E(X) \mid U_q(X), D_r(X)\right] = 1$. Otherwise, if $r \in B$, given that $U_q(X)$ and $D_r(X)$ hold, any box containing exactly two red points of $S \cap X$ and the origin o, where one red point is not in $Y_{q,r}$, will contain q or r in the interior. Then

$$\Pr\left[E(X) \mid U_q(X), D_r(X)\right] = \Pr[E(Y_{q,r}) \mid U_q(Y_{q,r})].$$

Similar arguments are given in the next two cases.

Case 2: $0 < x(q) < x(r)$. We have

$$\Pr[E(X) \mid U_q(X), D_r(X)] = \Pr[E(X_q \cup \{q\}) \mid U_q(X_q \cup \{q\})].$$

Case 3: $0 < x(r) < x(q)$. If $r \in R$, then

$$\Pr[E(X) \mid U_q(X), D_r(X)] = \Pr[E(Z_{q,r} \cup \{r\}) \mid D_r(Z_{q,r} \cup \{r\})],$$

where $Z_{q,r} \subset X$ contains the points $p \in X$ such that $x(p) < x(r)$ and either $y(p) < y(r)$ or $y(p) > y(q)$. Note that the event $[E(Z_{q,r} \cup \{r\}) \mid D_r(Z_{q,r} \cup \{r\})]$ is symmetric to the event $[E(X) \mid U_q(X)]$, thus its probability can be computed similarly. Otherwise, if $r \in B$, we have

$$\Pr[E(X) \mid U_q(X), D_r(X)] = \Pr[E(Z_{q,r})].$$

Note that in the above recursive computation of $\Pr[E(X)]$, for $X = R \cup B$, there is a polynomial number of subsets X_q, $Y_{q,r}$, and $Z_{q,r}$; each of such subsets can be encoded in constant space (i.e., by using a constant number of coordinates). Then, we can use dynamic programming, with a polynomial-size table, to compute $\Pr[E(R \cap B)]$ in time polynomial in n. $\square$

4 Discussion and Open Problems

For fixed d, the maximum box problem for non-probabilistic points can be solved in polynomial time [4]. Then, generating a polynomial number of outcomes of the probabilistic point set, and computing a maximum box for each of them, can be used to estimate both the probability that the total weight of a maximum box is at least a given parameter, and its expectation; in overall polynomial time and with high probability of success. We have as future work to design polynomial-time algorithms to approximate both values with deterministic success.

For the case of red and blue points in the plane, there are several open problems: For example, to compute $\Pr[\mathsf{box}(S) \geq k]$ (even for $k = 3$) when the box is restricted to contain a fixed point. Other variants appear when the box is restricted to contain a given point as vertex, or to have some side contained in a given line. All these variants can be considered when all blue points have probability 1 and all red ones probability less than 1. For red and blue points in $d = 1$, both $\Pr[\mathsf{box}(S) \geq k]$ and $\mathbb{E}[\mathsf{box}(S)]$ can be solved in polynomial time.

Acknowledgements. L.E.C. and I.V. are supported by MTM2016-76272-R (AEI/FEDER, UE). L.E.C. is also supported by the Spanish Government under the FPU grant agreement FPU14/04705. P.P.L. is supported by CONICYT FONDECYT/Regular 1160543 (Chile). C.S. is supported by Gen. Cat. DGR 2014SGR46 and MINECO MTM2015-63791-R.

References

1. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Range-max queries on uncertain data. J. Comput. Syst. Sci. (2017)
2. Alliez, P., Devillers, O., Snoeyink, J.: Removing degeneracies by perturbing the problem or perturbing the world. Reliab. Comput. **6**(1), 61–79 (2000)
3. Backer, J., Keil, J.M.: The mono- and bichromatic empty rectangle and square problems in all dimensions. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 14–25. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12200-2_3
4. Barbay, J., Chan, T.M., Navarro, G., Pérez-Lantero, P.: Maximum-weight planar boxes in $O(n^2)$ time (and better). Inf. Process. Lett. **114**(8), 437–445 (2014)
5. Caraballo, L.E., Ochoa, C., Pérez-Lantero, P., Rojas-Ledesma, J.: Matching colored points with rectangles. J. Comb. Optim. **33**(2), 403–421 (2017)
6. Chan, T.M., Kamousi, P., Suri, S.: Stochastic minimum spanning trees in Euclidean spaces. In: SoCG 2011, pp. 65–74 (2011)
7. Cortés, C., Díaz-Báñez, J.M., Pérez-Lantero, P., Seara, C., Urrutia, J., Ventura, I.: Bichromatic separability with two boxes: a general approach. J. Algorithms **64**(2–3), 79–88 (2009)
8. Faliszewski, P., Hemaspaandra, L.: The complexity of power-index comparison. Theor. Comput. Sci. **410**(1), 101–107 (2009)
9. Feldman, D., Munteanu, A., Sohler, C.: Smallest enclosing ball for probabilistic data. In: SoCG 2014, pp. 214–223 (2014)
10. Fink, M., Hershberger, J., Kumar, N., Suri, S.: Hyperplane separability and convexity of probabilistic point sets. JCG **8**(2), 32–57 (2017)
11. Kamousi, P., Chan, T.M., Suri, S.: Closest pair and the post office problem for stochastic points. Comput. Geom. **47**(2), 214–223 (2014)
12. Pérez-Lantero, P.: Area and perimeter of the convex hull of stochastic points. Comput. J. **59**(8), 1144–1154 (2016)
13. Suri, S., Verbeek, K., Yıldız, H.: On the most likely convex hull of uncertain points. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 791–802. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40450-4_67
14. Vadhan, S.P.: The complexity of counting in sparse, regular, and planar graphs. SIAM J. Comput. **31**(2), 398–427 (2001)
15. Valiant, L.G.: Universality considerations in VLSI circuits. IEEE Trans. Comput. **100**(2), 135–140 (1981)

The Online Set Aggregation Problem

Rodrigo A. Carrasco[1(✉)], Kirk Pruhs[2], Cliff Stein[3], and José Verschae[4]

[1] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile
rax@uai.cl

[2] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA
kirk@cs.pitt.edu

[3] Department of Industrial Engineering and Operations Research, Columbia University, New York, NY, USA
cliff@ieor.columbia.edu

[4] Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile
jverschae@uc.cl

Abstract. We introduce the online *Set Aggregation Problem*, which is a natural generalization of the *Multi-Level Aggregation Problem*, which in turn generalizes the *TCP Acknowledgment Problem* and the *Joint Replenishment Problem*. We give a deterministic online algorithm, and show that its competitive ratio is logarithmic in the number of requests. We also give a matching lower bound on the competitive ratio of any randomized online algorithm.

Keywords: Online algorithms · Competitive analysis
Set aggregation · Multilevel aggregation

1 Introduction

Problem Statement: We introduce an online problem, which we call the *Set Aggregation Problem*. In this problem, a sequence R of requests arrives over time. We assume time is continuous. Each request $\rho \in R$ has an associated release time r_ρ when it arrives, and an associated waiting cost function $w_\rho(t)$ that specifies the waiting cost for this request if it is first serviced at time t. We assume that the online algorithm learns $w_\rho(t)$ at time r_ρ. We also assume that each $w_\rho(t)$ is non-decreasing, left continuous, and $\lim_{t\to\infty} w_\rho(t) = \infty$.

At any time t, the online algorithm can decide to service any subset S of the previously released requests. Thus, a schedule for this instance is a sequence

R. A. Carrasco—Supported in part by Fondecyt Project Nr. 1151098.
K. Pruhs—Supported in part by NSF grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.
C. Stein—Supported in part by NSF grant CCF-1421161.
J. Verschae—Supported in part by Nucleo Milenio Información y Coordinación en Redes ICM/FIC CODIGO RC130003, and Fondecyt Project Nr. 11140579.

$(S_1, t_1), (S_2, t_2), \ldots, (S_k, t_k)$, where the S_i's are sets of requests and the t_i's are the times that these sets were serviced. We will implicitly restrict our attention to feasible schedules, which are those for which every request is serviced, i.e. for all $\rho \in R$, exists an (S_i, t_i) in the resulting sequence where $\rho \in S_i$ and $t_i \geq r_\rho$.

We assume that there is a time-invariant service cost function $\mathrm{C}(S)$ that specifies the cost for servicing a set S of the requests. Without any real loss of generality, we will generally assume that $\mathrm{C}(S)$ is monotone in the sense that adding requests to S can not decrease $\mathrm{C}(S)$. We will postpone the discussion on how the online algorithm learns $\mathrm{C}(S)$ until we state our results, but essentially our lower bound holds even if the online algorithm knows $\mathrm{C}(S)$ a priori, and our upper bound holds even if $\mathrm{C}(S)$ is only known for subsets S of released requests.

The online algorithm then incurs in two types of costs, a waiting cost and a service cost. At time t_i the algorithm incurs a waiting cost of $\mathrm{W}_t(S_i) = \sum_{\rho \in S_i} w_\rho(t)$ for servicing set S_i, which is just the aggregate waiting cost of the serviced requests. At the same time t_i the algorithm also incurs in a service cost of $\mathrm{C}(S_i)$, and the total service cost the algorithm incurs is $\sum_{i=1}^{k} \mathrm{C}(S_i)$. The total cost associated with schedule $(S_1, t_1), (S_2, t_2), \ldots, (S_k, t_k)$ is therefore $\sum_{i=1}^{k} \mathrm{C}(S_i) + \sum_{i=1}^{k} \mathrm{W}_t(S_i)$, and the objective is to find the schedule that minimizes the total cost.

In the deadline version of the set aggregation problem, the waiting cost of each request is zero until a specified deadline for that request, after which the waiting cost becomes infinite.

Most Relevant Background: Our main motivation for introducing the Set Aggregation Problem is that it is a natural generalization of the *Multi-Level Aggregation Problem* (MLAP), which was introduced by [1]. In the MLAP, the requests are vertices in an a priori known tree T, and the edges (and/or vertices) of T have associated costs. Further, a set S of requests has service cost equal to the minimum cost subtree of T that contains the root of T and contains all of the requests in S. The motivation for [1] to introduce the MLAP is that it generalizes both the well-known TCP Acknowledgment Problem (TCPAP), and the well-known Joint Replenishment Problem (JRP), both of which correspond to special cases of the MLAP in which the tree T has constant height. In [1] the authors gave an online algorithm for the Multi-Level Aggregation Problem, and showed that the competitive ratio of the algorithm is $O(d^4 2^d)$, where d is the height of T. It is an open question whether constant competitiveness is achievable in the MLAP, and only a constant lower bound is known. In [2] the authors gave an $O(d)$-approximation for the deadline version of the problem.

Our Results: The Set Aggregation Problem allows us to study how critical is the restriction that service costs come from an underlying tree. More specifically it is natural to ask:

– Can constant competitiveness be achieved if there are no restrictions on service costs (other than monotonicity)?
– And if not, what is the best achievable competitive ratio?

In this paper we answer these two questions. We first show, in Sect. 2, that the competitive ratio of every algorithm (deterministic or randomized) is at least logarithmic in the number of requests for the deadline version of the set aggregation problem. This lower bound holds against an online algorithm that a priori knows the domain $\mathcal{R}$ of all possible requests that might arrive, and the service cost $C(S)$ for each possible subset S of $\mathcal{R}$. Thus, we can conclude that if a constant competitive algorithm exists for the Multi-Level Aggregation Problem, then the analysis must use the fact that the service costs can not be arbitrary.

Intuitively, the requests in our lower bound instance are each associated with a node in a full binary tree T. There are $n/2^d$ requests associated with a node of depth d in the tree. The lifetime of the requests inherit the same laminar structure of the tree– the lifetime of the n requests associated with the root of T is $[0, n]$, the lifetime of the $n/2$ requests associated with the left child of the root is $[0, n/2]$, and the lifetime of the $n/2$ requests associated with the right child of the root is $[n/2, \mathrm{n}]$. The set costs are defined so that there is clearly no benefit to include more than one request associated with each node in any serviced set. Then the requests in the subtree rooted at the left child of the root aggregate with all requests associated with the root, but the only requests associated with the root that aggregate with requests associated with the right child of the root are those that the algorithm serviced during the first half of these request's lifetime. So the algorithm incurs an unnecessary incremental cost for each set serviced during $[n/2, n]$. The instance recursively applies this same idea lower down in T.

To complement this lower bound, in Sect. 3 we give a deterministic online algorithm RetrospectiveCover for the set aggregation, and show the competitive ratio of this algorithm is logarithmic in the number of requests. The algorithm only needs to know the service cost $C(S)$ for subsets S of requests that are released but unserviced.

Let us give a brief (necessarily simplified) overview of, and intuition for, the RetrospectiveCover algorithm. Define a proactive schedule to be one in which the total waiting cost of every serviced set is at most its service cost. The algorithm maintains a lower bound $\mathrm{LB}(t)$ for the least possible service cost incurred by any proactive schedule up until the current time t. Let u be a time where $\mathrm{LB}(u) = 2^k$ and let v be the future time where $\mathrm{LB}(v) = 2^k$, assuming no more requests are released. Intuitively, at time u the sets in the proactive schedule associated with $\mathrm{LB}(v)$ are serviced, and then a recursive call is made to handle requests that arrive until the time w when $\mathrm{LB}(w) = 2^{k+1}$. Note that due to the release of new requests it may be that w is earlier than v.

Computing the state of the optimum at the current time, and moving to that state, is a classic technique in online algorithms, and is often called the Retrospective algorithm [3]. For example, the Retrospective algorithm is optimally competitive for the online metric matching problem [4,5]. Intuitively the RetrospectiveCover algorithm is a generalization of the Retrospective algorithm, that computes the state of optimal at many carefully-selected times, for many different carefully-selected sub-instances, and then moves to a state that somehow covers/combines all of these optimal states.

This RetrospectiveCover algorithmic design technique seems relatively general, and at least plausibly applicable to other problems.

Within the context of the Multi-Level Aggregation problem, our upper bound on the achievable competitive ratio is incomparable to the upper bound obtained in [1]. The upper bound obtained in [1] is better if $d^r 2^d$ is asymptotically less than the logarithm of the number of requests, otherwise the upper bound that we obtain is better. As a caveat, computing the lower bound used by our algorithm is definitely NP-hard, and its not clear to us how to even obtain a polynomial-time offline $O(\log |R|)$-approximation algorithm. Techniques used in the prior literature on offline algorithms for TCPAP and JRP do not seem to be applicable.

Further Background: As mentioned above, the Set Aggregation Problem generalizes the TCPAP and the JRP. More generally, we can think of the case of the Set Aggregation Problem where the set relationship forms a tree. More precisely, if we include an edge for every pair of sets S_1 and S_2, where $S_1 \subset S_2$ and there is no set S_3 such that $S_1 \subset S_3 \subset S_2$, the family of sets S_i is laminar and the resulting graph is a tree. In [6], this problem is referred to as the Multi-Level Aggregation Problem. If the tree is of height one (a root and leaves), we obtain the TCPAP. The offline version of the TCPAP on n requests can be solved exactly in $O(n \log n)$ time [7]. The best deterministic approximation has competitive ratio 2 [8], and the best randomized is $e/(e-1)$ [9]. We note that the TCPAP is equivalent to the Lot Sizing Problem that has been studied in the operations research literature since the 1950s.

If the tree has two levels, we obtain the Joint Replenishment Problem. The best offline approximation is 1.791 [10] and the best competitive ratio is 3, via a primal dual-algorithm [11]. There is also a deadline version of the JRP where there is no cost for waiting but each request must be satisfied before its deadline. This problem is a special case of a general cost function and has an approximation ratio of 1.574 [12] and online competitive ratio of 2 [10].

For general trees of height D, we obtain the Multi-Level Aggregation Problem. This more general problem has several applications in computing, including protocols for aggregating control messages [13,14], energy efficient data aggregation and fusion in sensor networks [15,16], and message aggregation in organizational hierarchies [17]. There are also applications in lot sizing problems [18–20]. For the deadline version of the MLAP, there is an offline 2-approximation algorithm [21]. In unpublished work, Pedrosa [22] showed how to adapt an algorithm of Levi et al. [23] for the Multistage Assembly Problem to obtain a $2+\epsilon$ approximation algorithm for the MLAP with general waiting cost functions. For the online case, there is no constant competitive algorithm known. In [1], the authors give a $O(d^4 2^d)$ competitive algorithm for trees of height d, and a somewhat better bound of $O(d^2 2^d)$ for the deadline version. Their algorithms use a reduction from general trees to trees of exponentially decreasing weights as one goes down the tree, and therefore relies heavily on the tree structure. Building on this, [2] give an $O(d)$-competitive algorithm for the deadline version.

2 The Lower Bound

We prove a $\Omega(\log |R|)$ lower bound on the competitive ratio of any deterministic online algorithm for the deadline version of the Set Aggregation Problem.

Instance Construction. Conceptually the requests are partitioned such that each request is associated with a node in a full n-node binary tree T. The root of T has depth 0 and height $\lg n$. A leaf of T has depth $\lg n$ and height 0. A node of height h in T will have 2^h requests associated with it. Thus a leaf in T has one associated request, and the root of T has n requests associated with it. Hence there are n requests per level and $n(1 + \lg n)$ in total. The lifetime of a request associated with the k^{th} node at depth d (so $k \in \{1, 2, \ldots, 2^d\}$) in the tree is $[(k-1)n/2^d, kn/2^d)$. So the lifetime of all requests associated with the same node are the same, the lifetimes of the requests inherit the same laminar structure from T, and the lifetimes of the requests associated with nodes at a particular level of the tree cover the time interval $[0, n)$. Let R_x be the requests associated with a node x in T, and T_x be the requests associated with nodes in the subtree rooted at x in T. We say a collection S of requests is *sparse* if it does not contain two requests associated with nodes of the same depth in T.

We now proceed to describe the service costs. To do so we will split each set R_x into two sets of equal cardinality by defining a set $U_x \subseteq R_x$ with $|U_x| = |R_x|/2$. The specific requests that belong to U_x will be decided online by the adversary depending on partial outputs of the algorithm. Let us consider a set of requests S. If S is ever serviced by any algorithm at a given point in time $t \in (0, n)$, the laminar structure of the lifetimes implies that the requests correspond to a subset of nodes in a path P of T with endpoints at the root and some leaf v of T, where the lifetime of v contains t. Otherwise, we can define the cost of S arbitrarily, e.g., as ∞. We first define the cost of a set S that is sparse. The cost of S is defined inductively by walking up the path P. We say that a request r *aggregates* with a set S if the service cost of $S \cup \{r\}$ is the same as the service cost of S. If r does not aggregate with S then the service cost of $S \cup \{r\}$ is the service cost of S plus one. Finally, we say that r is *not compatible* with S if the cost of $S \cup \{r\}$ is $+\infty$. If S is a sparse set of requests in R_x where x is a leaf, then $|S| = 1$ and its service cost is also one. Consider now a sparse set $S \neq \emptyset$ with requests belonging to T_y for some y. Let x be the parent of y and r a request corresponding to x. Note that x is not a node associated to the requests in S. We have three cases:

- If $S \cap R_y = \emptyset$, then r is not compatible with S.
- Otherwise if y is a left-child of x, then r always aggregates with S.
- Otherwise if y is a right-child of x, then r aggregates with S when $r \notin U_x$, and it does not aggregate with S if $r \in U_x$.

Notice that, using this definition inductively, we have that the cost of $S \neq \emptyset$ is either infinity or an integer between 1 and the height of y plus one (inclusive)[1]. Also, the first condition implies that for a sparse set S to have finite cost there must exists a unique path P from a leaf to node x such that exactly one request in S belongs to R_v for each node v of P.

If S is an arbitrary set of requests (always corresponding to a leaf-to-root path P), then S can be decomposed into several sparse sets. We define the cost of S as the minimum over all possible decomposition, of the sum of the costs of the sparse sets in the decomposition. In this way, any solution can be converted to a solution of the same cost where each serviced set is sparse. Hence, we can restrict ourselves to consider only sparse sets.

Adversarial Strategy. We now explain how to choose the requests in U_x for each $x \in T$. Let (a_x, c_x) be the lifetime of the requests associated with x, and b_x the midpoint of (a_x, c_x). Let the left and right children of x be $\ell(x)$ and $r(x)$, respectively. Let S_x be the requests in R_x that the online algorithm has serviced by time b_x. If $|S_x| \leq |R_x|/2$ then let U_x be an arbitrary subset of $|R_x|/2$ requests from $R_x - S_x$. If $|S_x| > |R_x|/2$ then let U_x contain all the requests in $R_x - S_x$ plus an arbitrary set of $|S_x| - |R_x|/2$ requests from S_x.

To see that this is a valid adversarial strategy, consider a node x in T, and let P be the path in T from the root to x, and $u_1, \ldots, u_k$ be the nodes in P whose right child is also in P. The requests associated with ancestors of x in T that requests in R_x will not aggregate with are affected by the sets $U_{u_1}, \ldots U_{u_k}$, which are all known by time a_x.

Competitiveness Analysis. It is not hard to see that the optimum has cost at most n. We can construct a solution where each serviced set is sparse as follows. For each node x in T, any one request in U_x is serviced at each time in (a_x, b_x), and any one request from $R_x - U_x$ is serviced at each time in (b_x, c_x). We can now construct a solution in which each serviced set S corresponds to each different leaf-to-root path, where each node in the path corresponds to exactly one request in S. Each set S can me made to have a service cost of one. Since there are n leaves, thus n leaf-to-root paths, the constructed solution will have cost n.

Now consider the cost to the online algorithm for requests in R_x. We say that the *incremental cost at* x is how much more the online algorithm would pay for servicing the requests associated to nodes in T_x than it would pay for servicing requests in $T_x - R_x$ if requests in R_x were deleted from any (sparse) set S serviced by the online algorithm. First consider the case that $|S_x| \leq |R_x|/2$. In this case at time b_x the online algorithm has $|R_x|/2$ unserviced requests in U_x that do not aggregate with any requests in $T_{r(x)}$. Each such request can be only serviced together with a set $S \subseteq T_{r(x)}$, and thus each request in U_x imply an increase of 1 in the cost. Then the incremental cost at x is at least $|R_x|/2$.

[1] We remark that this definition does not yield monotone service costs. However this does not cause any trouble. Indeed, if two sets $S_1 \subseteq S_2$ fulfill $C(S_2) < C(S_1)$, then the algorithm can simply serve S_2 instead of S_1 without increasing its cost and without affecting feasibility. Hence, any instance with service cost C can be turned to an equivalent instance with non-decreasing service cost $C'(S) = \min_{T \supseteq S} C(T)$.

Now let us assume that $|S_x| > |R_x|/2$. Consider requests in S_x, which are serviced during the time period (a_x, b_x). Indeed, a request in S_x can be compatible with at most $|R_{\ell(x)}| = |R_x|/2$ many sparse sets within $R_{\ell(x)}$ (one for each request in $R_{\ell(x)}$). Hence, in this case the online algorithm incurs an incremental cost of at least $|S_x| - |R_x|/2$ for the requests in S_x. Request in $R_x - S_x = U_x$ all do not aggregate (or are incompatible) with sparse sets in $T_{r(x)}$, and hence the incur a cost of $|R_x| - |S_x|$. Thus in both cases the total the incremental cost at x is at least $|R_x|/2$. As there are $n \log n$ requests in total, the online algorithms pays at least $(n \log n)/2$. Thus we have shown a lower bound of $\Omega(\log n)$ on the competitive ratio.

Note that one can apply Yao's technique, where the identity of the requests in U_x are selected uniformly at random from the requests in R_x, to get an $\Omega(\log n)$ lower bound on the competitive ratio of any randomized online algorithm.

Theorem 1. *Any randomized online algorithm for the deadline version of the online set aggregation problem is $\Omega(\log(|R|))$-competitive.*

3 The Upper Bound

In Subsect. 3.1 we define a lower bound on the optimum used within the online algorithm, and observe some relatively straightforward properties of this lower bound. In Subsect. 3.2 we state the online algorithm. In Subsect. 3.3 we show that the online algorithm is $O(\log |R|)$ competitive.

3.1 Lower Bound on the Optimal Solution

We will simplify our lower bound and algorithm by restricting our attention to sets whose waiting cost is at most their service cost. By doing so, we will be able to focus only on service cost.

Definition 1. *We say that a set S of requests is* violated *at time t in a schedule if time t $\mathrm{W}_t(S) > C(S)$. A feasible schedule is* proactive *if it does not contain any violated set.*

We will also use in our algorithm a lower bound on the service cost of schedules for subsets of the input in subintervals of time. As mentioned in Sect. 1, computing the lower bound is NP-hard, but it is critical to guide our online algorithm.

Definition 2. *The lower bound $LB^-(s, t, d)$ is the minimum cost over all proactive schedules $\mathcal{Z}$ for the requests released in the time interval (s, t), of the total service cost incurred in $\mathcal{Z}$ during the time period (s, d). Let $LB^+(s, t, d)$ be the minimum over all proactive schedules $\mathcal{Z}$ for the requests released in the time interval (s, t), of the total service cost incurred in $\mathcal{Z}$ during the time period $(s, d]$. Polymorphically we will also use $LB^-(s, t, d)$ (resp. $LB^+(s, t, d)$) to denote the sets serviced within (s, d) (resp. $(s, d]$) by the proactive schedules that attains the minimum.*

The difference between $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ is that service cost incurred at time d are included in $\mathrm{LB}^+(s,t,d)$, but not in $\mathrm{LB}^-(s,t,d)$. Notice that the values of $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ do not depend on future requests, and thus can be computed at time t by an online algorithm.

Lemma 1. *There exists a proactive schedule whose objective value is at most twice optimal.*

Lemma 2. *The value $LB^-(s,t,d)$ and $LB^+(s,t,d)$ are monotone on the set of requests, that is, adding more requests between times s and t can not decrease either. Moreover, $LB^-(s,t,d)$ and $LB^+(s,t,d)$ are non-decreasing as a function of t and as a function of d, for any $s \le t \le d$.*

The proof for Lemmas 1 and 2 are in Sect. A.

3.2 Algorithm Design

We now give our algorithm RetrospectiveCover, which is executed at each time t. Although we understand that this is nonstandard, we believe that most intuitive way to conceptualize our algorithm is to think of the algorithm as executing several concurrent processes. The active processes are numbered $1, 2, \ldots, a$. Each process i maintains a start time $s[i]$. A process i reaches a new milestone at time t if the value of $\mathrm{LB}^+(s[i],t,t)$ is at least $2 \cdot \mathrm{LB}^+(s[i],m[i],m[i])$, where $m[i]$ is the time of the last milestone for process i. When a milestone for process i is reached at time t, each higher numbered process ℓ services the sets in $\mathrm{LB}^-(s[\ell],t,t)$, and then terminates. Process i then services the sets in $\mathrm{LB}^-(s[i],t,d[i])$, where $d[i]$ is the earliest time after t where $\mathrm{LB}^+(s[i],t,d[i]) \ge 2 \cdot \mathrm{LB}^+(s[i],t,t)$. Process i then starts a process $i+1$ with start time of t.

We give pseudocode for our algorithm RetrospectiveCover to explain how various corner cases are handled, and to aide readers who don't want to think about the algorithm in terms of concurrent processes. RetrospectiveCover uses a subroutine CheckMilestone that checks, for a process i, whether it has reached a milestone at the current time t. RetrospectiveCover is initialized by setting $a = 1$, $s[1] = 0$ and $m[1] = s[1]$.

```
CheckMilestone($i, t$)
1  if $m[i] = s[i]$ then
2  // Process $i$ has not seen its first milestone yet
3      if $\mathrm{LB}^+(s[i],t,t) > 0$ then
4          return true
5      else return false
6  else
7      if $\mathrm{LB}^+(s[i],t,t) \ge 2 \cdot \mathrm{LB}^+(s[i],m[i],m[i])$ then
8          return true
9      else return false
```

```
RetrospectiveCover($t$)
1  for $i = 1$ to $a$
2      if CheckMilestone($i, t$) then
3          for $\ell = a$ downto $i + 1$
4              if any request has arrived since time $m[\ell]$ then
5                  service every set in $\text{LB}^-(s[\ell], t, t)$
                   // Intuitively process $\ell$ now terminates
6          // The analysis will prove in an inductive invariant on the state
           at this point
7          Let $d[i]$ be the earliest time after $t$ where
           $\text{LB}^+(s[i], t, d[i]) \geq 2 \cdot \text{LB}^+(s[i], t, t)$
             if such a time exists, and $d[i]$ is infinite otherwise.
8          Service the sets in $\text{LB}^-(s[i], t, d[i])$
9          $m[i] = t$
10         $a = i + 1$; $s[a] = t$; $m[a] = s[a]$; return
```

3.3 Algorithm Analysis

The first part of this subsection is devoted to proving Lemma 4, which is the key lemma, and states that the service cost incurred by RetrospectiveCover is within a logarithmic factor of optimal. Next, we show how this readily implies that RetrospectiveCover is $O(\log |R|)$-competitive.

We first make some simplifying assumptions, and then prove one simple property of RetrospectiveCover. Without loss of generality, we can assume that no two requests are released at the same time, and that no requests are released exactly at any milestone (one can sequentialize the releases arbitrarily followed by the unique milestone). Similarly, without loss of generality we can assume that each waiting function $w_\rho(t)$ is a continuous function of t. Finally, we can assume without loss of generality that the initial waiting time is 0, that is $w_\rho(r_\rho) = 0$. With this we can state the following Lemma, which is proved in Sect. A.

Lemma 3. *Assume that* RetrospectiveCover *just computed a new deadline $d[i]$ in line 7. Then process i will either reach a new milestone or be terminated by time $d[i]$.*

3.3.1 The Key Induction Argument

Lemma 4. *Let $c = 6$. Consider a point of time when* RetrospectiveCover *is at line 6. Let Y be the number of requests that arrive during the time interval $(s[i], t)$. Then the service cost incurred by* RetrospectiveCover *up until this point is at most $c(\lg Y) LB^+(s[i], t, t)$.*

The rest of this subsubsection is devoted to proving Lemma 4 by induction on the number of times that line 6 in RetrospectiveCover is invoked. So consider an arbitrary time that RetrospectiveCover is at line 6. We now need to introduce some more notation. (See Fig. 1 for an illustration of the various concepts and notation). Let $u_{-1} = s[i]$, $u_0 = m[i]$, y_0 be the number of

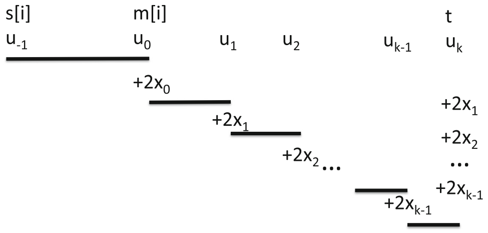


Fig. 1. An illustration of the generic situation, and when costs are incurred by RetrospectiveCover. The terms in the diagonal depict the cost incurred in line 8 and the terms on the right terms in line 5.

requests that arrive during (s, u_0), and $x_0 = \mathrm{LB}^+(s[i], m[i], m[i])$. Let $k = a - i$. Then for $j \in \{1, 2, \ldots, k\}$, let $u_{j-1} = s[i+j] = m[i+j-1]$, $u_k = t$, let y_j be the number of requests that arrive during (u_{j-1}, u_j), and $x_j = \mathrm{LB}^+(u_{j-1}, u_j, u_j)$. Also, for notational convenience let $z = \mathrm{LB}^+(s[i], t, t)$.

Before making our inductive argument, we need to detour slightly. Lemma 5 gives an inequality on service costs that will be useful in our analysis, and the proof is given in Sect. A.

Lemma 5. *It holds that* $\sum_{j=1}^{k-1} x_j \leq x_0$.

We are now ready to make our inductive argument. We start with the base case. The first milestone occurs at the first time t where $\mathrm{LB}^+(0, t, t)$ is positive. When this occurs, RetrospectiveCover has incurred no service cost until this point because $\mathrm{LB}^-(0, t, t) = 0$. Note that if $Y = 1$ or $k = 0$, then again RetrospectiveCover has incurred no service cost until this point because $\mathrm{LB}^-(0, t, t) = 0$. So assume from now on that $Y \geq 2$ and $k \geq 1$.

Now consider the case that $k = 1$. In this case, process i incurred a cost of at most $2x_0$ when line 8 is invoked at time u_0, and as process $i+1$ did not reach its first milestone (which would have caused process $i+2$ to start), no additional cost is incurred at time t. Thus to establish the induction, it is sufficient to show that $2x_0 + cx_0 \lg y_0 \leq cz \lg Y$.

Normalizing costs so that $x_0 = 1$, and using the fact that $z \geq 2x_0$ (since t is the next milestone), it is sufficient to show that $4y_0^c \leq Y^{2c}$. Using the fact that $y_0 \leq Y$, it is sufficient to show $4Y^c \leq Y^{2c}$. This holds as $Y \geq 2$ and $c = 6$.

Now consider the case that $k = 2$. In this case, process i incurred a cost of at most $2x_0$ when line 8 is invoked at time u_0, process $i+1$ incurs a cost of at most $2x_1$ when line 8 is invoked at time u_1 and at most $2x_1$ when line 5 is invoked at time t, and process $i+2$ incurs no cost at time t. Thus to establish the induction, it is sufficient to show that $2x_0 + 4x_1 + cx_0 \lg y_0 + cx_1 \lg y_1 \leq cz \lg Y$.

Normalizing costs so that $x_0 = 1$, and using the fact that $z \geq 2$, $x_1 \leq x_0$, and $y_0 + y_1 \leq Y$, it is sufficient to show that: $64 y_0^c y_1^c \leq (y_0 + y_1)^{2c}$, which holds by the binomial theorem as $64 \leq \binom{2c}{c} = \binom{12}{6}$.

For the remainder of the proof, we assume $k \geq 3$. At time u_j, for $0 \leq j \leq k-1$, process $i+j$ incurs a cost at most $2x_j$ when line 8 is invoked. At time t process $i+j$, for $1 \leq j \leq k-1$, incurs a cost of at most $2x_j$ when line 5 is invoked. Indeed, the algorithm pays $\mathrm{LB}^-(u_{j-1}, t, t)$, which must be less than $2\mathrm{LB}^+(u_{j-1}, u_j, u_j)$, otherwise process $i+j$ would have hit a milestone within (u_j, t). Similarly, at time t point process $i+k=a$ does not incur any cost, as otherwise process a would have hit a milestone before time t. Thus to establish the induction, we need to show that $2x_0 + 4\sum_{j=1}^{k-1} x_j + c\sum_{j=0}^{k-1} x_j \lg y_j \leq cz \lg Y$.

Note that our induction is using the fact that the cost for RetrospectiveCover during a time interval (u_{j-1}, u_j) is identical to the cost of RetrospectiveCover on the subinstance of requests that are released during (u_{j-1}, u_j), essentially because RetrospectiveCover can be viewed as a recursive algorithm. We now normalize costs so that $x_0 = 1$. Note that by Lemma 5, $\sum_{j=1}^{k-1} x_j \leq 1$, and $z \geq 2$ (or t wouldn't be the next milestone). Thus it is sufficient to show that $64\prod_{j=0}^{k-1} y_j^{cx_j} \leq Y^{2c}$.

We now claim that the left hand side of this inequality is maximized, subject to the constraint that $\sum_{j=0}^{k-1} y_j \leq Y$, when each $y_j = x_j Y/X$, where $X = \sum_{j=0}^{k-1} x_j$ (this can be shown using the method of Lagrangian multipliers). Thus it is sufficient to show that $64\prod_{j=0}^{k-1}(Yx_j/X)^{cx_j} \leq Y^{2c}$, which is equivalent to $64Y^{c(X-2)}\prod_{j=0}^{k-1}(x_j)^{cx_j} \leq X^{cX}$.

Since $X \leq 2$ and $Y \geq 2$, the value $Y^{c(X-2)}$ is maximized when $Y = 2$. Thus it suffices to show that $64 \cdot 2^{c(X-2)}\prod_{j=0}^{k-1}(x_j)^{cx_j} \leq X^{cX}$. Because $x_0 = 1$, it is sufficient to show that $64 \cdot 2^{c(X-2)}\prod_{j=1}^{k-1}(x_j)^{cx_j} \leq X^{cX}$.

Using again the method of Lagrangian multipliers, we have that the maximum of the left hand side of this inequality, subject to $\sum_{j=1}^{k-1} x_j = X-1$, is reached when all x_j are equal. Hence it suffices show that $64 \cdot 2^{c(X-2)}\left(\frac{X-1}{k-1}\right)^{cX} \leq X^{cX}$. Since $((X-1)/X)^{cX}$ is clearly between 0 and 1, it is sufficient to show that $2^{6-2c} \cdot 2^{cX} \leq (k-1)^{cX}$, which, by simple algebra, is true for $c = 6$ and when $k \geq 3$.

The Rest of the Analysis. Lemma 6, which is proved in Sect. A, shows that because the algorithm is trying to mimic proactive schedules, it will be the case that the waiting cost for the algorithm is at most twice its service cost. Finally in Theorem 2, also proved in Sect. A, we conclude that these lemmas imply that our algorithm is $O(\log|R|)$-competitive.

Lemma 6. *For any set S serviced at time t in the schedule produced by the algorithm* RetrospectiveCover, *it will be the case that* $\mathrm{W}_t(S) \leq 2\,\mathrm{C}(S)$.

Theorem 2. *The* RetrospectiveCover *algorithm is* $O(\log|R|)$*-competitive.*

4 Conclusion

Another possible way to generalize the multilevel aggregation problem is to assume that the domain $\mathcal{R}$ of possible requests (perhaps it is useful to think of $\mathcal{R}$ as "types" of possible requests), and the service cost $C(S)$ for every subset S of $\mathcal{R}$, is known to the online algorithm a priori, and then consider the competitive ratio as a function of $|\mathcal{R}|$. Our lower bound instance shows that the optimal competitive ratio is $\Omega(\log\log|\mathcal{R}|)$. Its not immediately clear to us how to upper bound the competitiveness of our algorithm RetrospectiveCover in terms of $|\mathcal{R}|$, or how to design an algorithm where such an analysis is natural. So, determining the optimal competitive ratio as a function of $|\mathcal{R}|$ seems like a reasonable interesting open problem.

A Detailed Proofs

In this section we detail some of the proofs from Sect. 3.

Lemma 1. There exists a proactive schedule whose objective value is at most twice optimal.

Proof. We show how to iteratively transform an arbitrary schedule $\mathcal{Z}$ into a proactive schedule in such a way that the total cost at most doubles. Let t be the next time in $\mathcal{Z}$ when there is an unserved set S, with the property that the waiting time for S, infinitesimally after t, is greater than the service cost for S. We then add the set S at time t to $\mathcal{Z}$. The service cost of this set is at most the total waiting time of the requests that it serves. Thus the total service cost of the final schedule is at most the waiting time in the original $\mathcal{Z}'$. Further the transformation can only decrease the total waiting cost, since the requests in S are being served no later than they were originally.

Lemma 2. The value $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ are monotone on the set of requests, that is, adding more requests between times s and t can not decrease either. Moreover, $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ are non-decreasing as a function of t and as a function of d, for any $s \le t \le d$.

Proof. Any schedule that is proactive for the larger set of requests is also proactive for the smaller set of requests, because the waiting time of the requests serviced can not be more in the smaller set of requests than in the larger set of requests. The monotonicity on t follows directly from the monotonicity on the set of requests. The monotonicity on d is clear since for any $d \le d'$, a proactive solution up to time d' is also proactive up to time d.

Lemma 3. Assume that RetrospectiveCover just computed a new deadline $d[i]$ in line 7. Then process i will either reach a new milestone or be terminated by time $d[i]$.

Proof. If $d[i]$ is infinite, then this is obvious, so assume otherwise. If process i is terminated before time $d[i]$ then this is obvious, so assume otherwise. If no requests arrive during the time interval $(m[i], d[i])$ then process i will reach a new milestone exactly at time $d[i]$ by the definition of milestones. If requests arrive before $d[i]$ then the claim follows by the monotonicity of LB^+, see Lemma 2.

Lemma 5. It holds that $\sum_{j=1}^{k-1} x_j \le x_0$.

Proof. First notice that $\mathrm{LB}^{-}(u_{-1}, t, t) \leq 2\mathrm{LB}^{+}(u_{-1}, u_0, u_0) = 2x_0$, since otherwise process i would have hit a milestone within the interval (u_0, t). Then, it suffices to show that $\sum_{j=0}^{k-1} x_j \leq \mathrm{LB}^{-}(u_{-1}, t, t)$. To show this last bound, fix $j \in \{0, \dots, k-1\}$ and consider a proactive schedule $\mathcal{Z}$ for requests arriving in (u_{-1}, t). Within each interval (u_{j-1}, u_j), the solution is proactive when considering requests with release date in $(u_{j-1}, u_j]$, and hence the serving cost of the requests served by $\mathcal{Z}$ within this interval is at most $\mathrm{LB}^{+}(u_{j-1}, u_{j-1}, u_j)$. We remark that this is also true for $j = k-1$ as $u_{k-1} < t$. Taking $\mathcal{Z}$ minimizing the service cost within (u_{-1}, t), we obtain the required bound, $\sum_{j=0}^{k-1} x_j \leq \mathrm{LB}^{-}(u_{-1}, t, t)$.

Lemma 6. For any set S serviced at time t in the schedule produced by the algorithm RetrospectiveCover, it will be the case that $\mathrm{W}_t(S) \leq 2\,\mathrm{C}(S)$.

Proof. Assume that a process i hit a milestone at time t. We adopt the notation from Subsubsect. 3.3.1 and illustrated in Fig. 1. Additionally let U_j be the requests that are released in the time interval (u_{j-1}, u_j) for $j \in \{0, 1, \dots, k\}$.

We now prove that the sets serviced in line 5 have waiting time at most twice the service cost. Also, we show that after serving such sets, the set $\cup_{h=k-j}^{k} U_h$ has no violated subset at time t, where $j = a - \ell$. We show this by induction on $j = a - \ell$. The base case is when $j = 0$ ($\ell = a$). There is no violated subset of U_k at time t since process $i + k = a$ did not hit a milestone before time t. Thus, no set serviced in $\mathrm{LB}^{-}(s[a], t, t)$ is violated. Obviously, after servicing such sets, U_k still has no violated subset. Now let us show the two properties for an arbitrary j. By induction hypothesis, the set $\cup_{h=k-(j-1)}^{k} U_h$ has no violated subset at time t. There is no violated subset of U_{k-j} at time t since the servicing of sets in $\mathrm{LB}^{-}(s[a-j], m[a-j], d[a-j]) = \mathrm{LB}^{-}(u_{k-j-1}, u_{k-j}, d[a-j])$ at time $m[a-j]$ guaranteed that U_{k-j} would not have any violated subsets until after time $d[a-j]$ and $t \leq d[a-j]$. Thus no set serviced in $\mathrm{LB}^{-}(s[a-j], t, t)$ in line 5 has waiting time at most twice the service cost. Further since $\mathrm{LB}^{-}(s[a-j], t, t) = \mathrm{LB}^{-}(u_{k-j-1}, t, t)$ is a proactive schedule, after serving sets in such solution the set $\cup_{h=k-j}^{k} U_h$ has no violated subset at time t.

Finally the same argument can be applied to the sets serviced in line 8 in RetrospectiveCover.

Theorem 2. The RetrospectiveCover algorithm is $O(\log |R|)$-competitive.

Proof. Applying Lemma 4 to the original process, and noting that the service cost in the final invocation of line 8 is at most twice the previous service costs, one obtains that the service cost for the algorithm is $O(\log |R|)$ times the lower bound. By Lemma 1 the lower bound is at most twice the optimal, and by Lemma 6 the waiting cost for requests serviced by the algorithm is at most the service cost of these requests. Finally the waiting cost of any unserviced requests is at most twice the service cost of the algorithm.

References

1. Bienkowski, M., Böhm, M., Byrka, J., Chrobak, M., Dürr, C., Folwarczný, L., Jez, L., Sgall, J., Thang, N.K., Veselý, P.: Online algorithms for multi-level aggregation. In: European Symposium on Algorithms, pp. 12:1–12:17 (2016)
2. Buchbinder, N., Feldman, M., Naor, J.S., Talmon, O.: O(depth)-competitive algorithm for online multi-level aggregation. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1235–1244 (2017)
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, New York (1998)
4. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
5. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theor. Comput. Sci. **127**(2), 255–267 (1994)
6. Bienkowski, M., Böhm, M., Byrka, J., Chrobak, M., Dürr, C., Folwarczný, L., Jeż, L., Sgall, J., Thang, N.K., Veselý, P.: Online algorithms for multi-level aggregation. arXiv preprint arXiv:1507.02378 (2015)
7. Aggarwal, A., Park, J.K.: Improved algorithms for economic lot sizing problems. Oper. Res. **41**, 549–571 (1993)
8. Dooly, D.R., Goldman, S.A., Scott, S.D.: On-line analysis of the TCP acknowledgment delay problem. J. ACM **48**(2), 243–273 (2001)
9. Karlin, A.R., Kenyon, C., Randall, D.: Dynamic TCP acknowledgement and other stories about $\mathrm{e}/(\mathrm{e}-1)$. Algorithmica **36**(3), 209–224 (2003)
10. Bienkowski, M., Byrka, J., Chrobak, M., Jeż, Ł., Nogneng, D., Sgall, J.: Better approximation bounds for the joint replenishment problem. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 42–54 (2014)
11. Buchbinder, N., Kimbrel, T., Levi, R., Makarychev, K., Sviridenko, M.: Online make-to-order joint replenishment model: primal-dual competitive algorithms. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 952–961 (2008)
12. Bienkowski, M., Byrka, J., Chrobak, M., Dobbs, N., Nowicki, T., Sviridenko, M., Świrszcz, G., Young, N.E.: Approximation algorithms for the joint replenishment problem with deadlines. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 135–147. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39206-1_12
13. Badrinath, B., Sudame, P.: Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. In: International Conference on Computer Communications and Networks, pp. 206–213 (2000)
14. Bortnikov, E., Cohen, R.: Schemes for scheduling of control messages by hierarchical protocols. In: Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 865–872 (1998)
15. Hu, F., Cao, X., May, C.: Optimized scheduling for data aggregation in wireless sensor networks. In: International Conference on Information Technology: Coding and Computing (ITCC 2005), vol. 2, pp. 557–561 (2005)
16. Yuan, W., Krishnamurthy, S., Tripathi, S.: Synchronization of multiple levels of data fusion in wireless sensor networks. In: Global Telecommunications Conference, vol. 1, pp. 221–225 (2003)
17. Papadimitriou, C.H.: Computational aspects of organization theory. In: Diaz, J., Serna, M. (eds.) ESA 1996. LNCS, vol. 1136, pp. 559–564. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61680-2_82

18. Crowston, W.B., Wagner, M.H.: Dynamic lot size models for multi-stage assembly systems. Manag. Sci. **20**(1), 14–21 (1973)
19. Kimms, A.: Multi-level lot sizing and scheduling: methods for capacitated, dynamic, and deterministic models. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-50162-3
20. Lambert, D.M., Cooper, M.C.: Issues in supply chain management. Ind. Mark. Manag. **29**(1), 65–83 (2000)
21. Becchetti, L., Marchetti-Spaccamela, A., Vitaletti, A., Korteweg, P., Skutella, M., Stougie, L.: Latency-constrained aggregation in sensor networks. ACM Trans. Algorithms **6**(1), 13:1–13:20 (2009)
22. Pedrosa, L.L.C.: Private communication (2013)
23. Levi, R., Roundy, R., Shmoys, D.B.: Primal-dual algorithms for deterministic inventory problems. Mathematics of Operations Research **31**(2), 267–284 (2006)

Agglomerative Clustering of Growing Squares

Thom Castermans[1](✉), Bettina Speckmann[1], Frank Staals[2], and Kevin Verbeek[1]

[1] TU Eindhoven, Eindhoven, The Netherlands
{t.h.a.castermans,b.speckmann,k.a.b.verbeek}@tue.nl
[2] Utrecht University, Utrecht, The Netherlands
f.staals@uu.nl

Abstract. We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. Consider a set of disjoint square glyphs on an interactive map. When the user zooms out, the glyphs grow in size relative to the map, possibly with different speeds. When two glyphs intersect, we wish to replace them by a new glyph that captures the information of the intersecting glyphs.

We present a fully dynamic kinetic data structure that maintains a set of n disjoint growing squares. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports queries in worst case $O(\log^3 n)$ time, and updates in $O(\log^7 n)$ amortized time. This leads to an $O(n\alpha(n)\log^7 n)$ time algorithm (where α is the inverse Ackermann function) to solve the agglomerative clustering problem, which is a significant improvement over the straightforward $O(n^2 \log n)$ time algorithm.

1 Introduction

We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. Specifically, *GlamMap*[1] [6] is a visual analytics tool for the eHumanities which allows the user to interactively explore datasets which contain metadata of a book collection. Each book is depicted by a square, color-coded by publication year, and placed on a map according to the location of its publisher. Overlapping squares are recursively aggregated into a larger glyph until all glyphs are disjoint. As the user zooms out, the glyphs "grow" relative to the map to remain legible. As a result, glyphs start to overlap and need to be merged into larger glyphs to keep the map clear and uncluttered. To allow the

The Netherlands Organisation for Scientific Research (NWO) is supporting T. Castermans (project number 314.99.117), B. Speckmann (project number 639.023.208), F. Staals (project number 612.001.651), and K. Verbeek (project number 639.021.541).

[1] http://glammap.net/glamdev/maps/1, best viewed in Chrome. GlamMap currently does not implement the algorithm described in this article.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 260–274, 2018.
https://doi.org/10.1007/978-3-319-77404-6_20

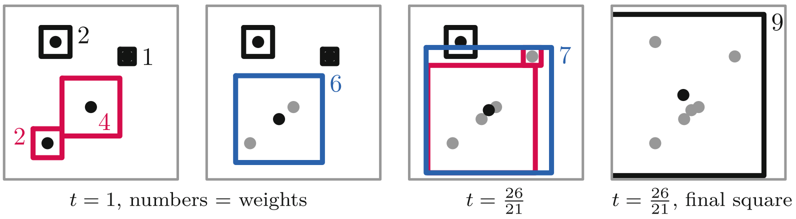


Fig. 1. Events for growing squares: intersecting squares in red, merged squares in blue. (Color figure online)

user to filter and browse real world data sets[2] at interactive speed we hence need an efficient agglomerative clustering algorithm for growing squares (glyphs).

Formal problem statement. Let P be a set of points in $\mathbb{R}^2$. Each point $p \in P$ has a positive weight p_w. Given a "time" parameter t, we interpret the points in P as squares. More specifically, let $\square_p(t)$ be the square centered at p with width tp_w. For ease of exposition we assume all point locations to be unique. With some abuse of notation we may refer to P as a set of squares rather than the set of center points of squares. Observe that initially, i.e. at $t = 0$, all squares in P are disjoint. As t increases, the squares in P grow, and hence they may start to intersect. When two squares $\square_p(t)$ and $\square_q(t)$ intersect at time t, we remove both p and q and replace them by a new point $z = \kappa p + (1 - \kappa)q$, with $\kappa = p_w/(p_w + q_w)$, of weight $z_w = p_w + q_w$ (see Fig. 1). Our goal is to compute the complete sequence of events where squares intersect and merge.

We present a fully dynamic data structure that uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time, which allows us to compute the agglomerative clustering for n squares in $O(n\alpha(n)\log^7 n)$ time. Here, α is the extremely slowly growing inverse Ackermann function. To the best of our knowledge, this is the first fully dynamic clustering algorithm which beats the straightforward $O(n^2 \log n)$ time bound.

Related Work. Funke et al. [7] introduced so-called "ball tournaments", a related, but simpler, problem, which is motivated by map labeling. Their input is a set of balls in $\mathbb{R}^d$ with an associated set of priorities. The balls grow linearly and whenever two balls touch, the ball with the lower priority is eliminated. The goal is to compute the elimination sequence efficiently. Bahrdt et al. [4] and Funke and Storandt [8] improved upon the initial results and presented bounds which depend on the ratio Δ of the largest to the smallest radius. Specifically, Funke and Storandt [8] show how to compute an elimination sequence for n balls in $O(n \log \Delta(\log n + \Delta^{d-1}))$ time in arbitrary dimensions and in $O(Cnpolylogn)$ time for $d = 2$, where C denotes the number of different radii. In our setting eliminations are not sufficient, since merged glyphs need to be re-inserted. Fur-

[2] For example, the catalogue of WorldCat contains more than 321 million library records at hundreds of thousands of distinct locations.

thermore, as opposed to typical map labeling problems where labels come in fixed sizes, our glyphs can vary by a factor of 10.000 or more.

Ahn et al. [2] very recently and independently developed the first subquadratic algorithms to compute elimination orders for ball tournaments. Their results apply to balls and boxes in two or higher dimensions. Specifically, for squares in two dimensions they can compute an elimination order in $O(n \log^4 n)$ time. Their results critically depend on the fact that they know the elimination priorities at the start of their algorithm and that they have to handle only deletions. Hence they do not have to run an explicit simulation of the growth process and can achieve their results by the clever use of advanced data structures. In contrast, we are handling the fully dynamic setting with both insertions and deletions, and without a specified set of priorities.

Our clustering problem combines both dynamic and kinetic aspects: squares grow, which is a restricted form of movement, and squares are both inserted and deleted. There are comparatively few papers which tackle dynamic kinetic problems. Alexandron et al. [3] present a dynamic and kinetic data structure for maintaining the convex hull of points moving in $\mathbb{R}^2$. Their data structure processes (in expectation) $O(n^2\beta_{s+2}(n)\log n)$ events in $O(\log^2 n)$ time each. Here, $\beta_s(n) = \lambda_s(n)/n$, and $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence on n symbols of order s. Agarwal et al. [1] present dynamic and kinetic data structures for maintaining the closest pair and all nearest neighbors. The expected number of events processed is roughly $O(n^2\beta_{s+2}(n) polylogn)$, each of which can be handled in $O(polylogn)$ expected time. Some of our ideas and constructions are similar in flavor to the structures presented in their paper.

Results and organization. We present a fully dynamic data structure that can maintain a set P of disjoint growing squares. Our data structure reports an *intersection event* at every time t when a square $\Box_q$ touches $\Box_p$ of a point $p \in P$ that *dominates* q. Here we say that a point p dominates q if and only if $q_x \leq p_x$ and $q_y \leq p_y$. We combine four of these data structures, one for each quadrant, to ensure that all squares in P remain disjoint. When our structure detects an intersection event we have to delete two or more of the squares and subsequently insert a new, merged, square. At any time, our data structure supports querying if a new square is disjoint from the ones in P (see Sect. 3.2), and inserting a new disjoint square or removing an existing square (see Sect. 3.3).

The crucial observation is that we can maintain the points $D(q)$ dominating q in an order so that a prefix of $D(q)$ will have their squares intersect the top side of $\Box_q$ first, and the remaining squares will intersect the right side of $\Box_q$ first. We formalize this in Sect. 2. We then present our data structure—essentially a pair of range trees interlinked with "linking certificates"—in Sect. 3. While our data structure is conceptually simple, the details are somewhat intricate. Our initial analysis shows that our data structure maintains $O(\log^6 n)$ certificates per square, which yields an $O(\log^7 n)$ amortized update time. This allows us to simulate the process of growing the squares in P—and thus solve the agglomerative glyph clustering problem—in $O(n\alpha(n)\log^7 n)$ time using $O(n \log^6 n)$ space.

In Sect. 4 we analyze the relation between canonical subsets in dominance queries. We show that for two range trees T^R and T^B in $\mathbb{R}^d$, the number of pairs of nodes $r \in T^R$ and $b \in T^B$ for which r occurs in the canonical subset of a dominance query defined by b and vice versa is only $O(n(\log n \log\log n)^2)$, where n is the total size of T^R and T^B. This implies that the number of linking certificates that our data structure maintains, as well as the total space used, is actually only $O(n(\log n \log\log n)^2)$. Since the linking certificates actually provide an efficient representation of all dominance relations between two point sets (or within a point set), we believe that this result is of independent interest.

All proofs omitted from this article can be found in the full version [5].

2 Geometric Properties

Let ℓ_q denote the bottom left vertex of a square $\square_q$, and let r_q denote the top right vertex of $\square_q$. Furthermore, let $D(q)$ denote the subset of points of P dominating q, and let $L(q) = \{\ell_p \mid p \in D(q)\}$ denote the set of bottom left vertices of the squares of those points.

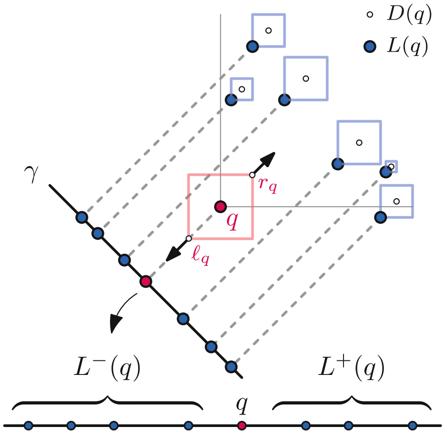


Fig. 2. The projection of the square centers and relevant corners onto line γ.

Observation 1. *Let $p \in D(q)$ be a point dominating point q. The squares $\square_q(t)$ and $\square_p(t)$ intersect at time t if and only if $r_q(t)$ dominates $\ell_p(t)$ at time t.*

Consider a line γ with slope minus one, project all points in $Z(t) = \{r_q(t)\} \cup L(q)(t)$, for some time t, onto γ, and order them from left to right. Observe that, since all points in Z move along lines with slope one, this order does not depend on the time t. Moreover, for any point p, we have $r_p(0) = \ell_p(0) = p$, so we can easily compute this order by projecting the centers of the squares onto γ and sorting them. Let $D^-(q)$ denote the (ordered) subset of $D(q)$ that occur before q in the order along γ, and let $D^+(q)$ denote the ordered subset of $D(q)$ that occur at or after q in the order along γ. We define $L^-(q)$ and $L^+(q)$ analogously (see Fig. 2).

Observation 2. *Let $p \in D(q)$ be a point dominating point q, and let t^* be the first time at which $r = r_q(t^*)$ dominates $\ell = \ell_p(t^*)$. We then have that*

- $\ell_x < r_x$ *and* $\ell_y = r_y$ *if and only if* $p \in D^-(q)$, *and*
- $\ell_x = r_x$ *and* $\ell_y \leq r_y$ *if and only if* $p \in D^+(q)$.

Observation 2 implies that the points p in $D^-(q)$ will start to intersect $\Box_q$ at some time t^* because the bottom left vertex ℓ_p of $\Box_p$ will enter $\Box_q$ through the top edge, whereas the bottom left vertex of the (squares of the) points in $D^+(q)$ will enter $\Box_q$ through the right edge. We thus obtain the following result.

Lemma 3. *Let t^* be the time that a square $\Box_p$ of a point $p \in D(q)$ touches $\Box_q$. We then have that*

(i) $r_q(t^*)_y = \ell_p(t^*)_y$, *and* $\ell_p(t^*)$ *is the point with minimum* y*-coordinate among the points in* $L^-(q)(t^*)$ *at time* t^**, if and only if* $p \in D^-(q)$*, and*
(ii) $r_q(t^*)_x = \ell_p(t^*)_x$, *and* $\ell_p(t^*)$ *is the point with minimum* x*-coordinate among the points in* $L^+(q)(t^*)$ *at time* t^**, otherwise (i.e. if and only if* $p \in D^+(q)$*).*

3 A Kinetic Data Structure for Growing Squares

In this section we present a data structure that can detect the first intersection among a dynamic set of disjoint growing squares. In particular, we describe a data structure that can detect intersections between all pairs of squares $\Box_p, \Box_q$ in P such that $p \in D^+(q)$. We build an analogous data structure for when $p \in D^-(q)$. This covers all intersections between pairs of squares $\Box_p, \Box_q$, where $p \in D(q)$. We then use four copies of these data structures, one for each quadrant, to detect the first intersection among all pairs of squares.

We describe the data structure itself in Sect. 3.1, and we briefly describe how to query it in Sect. 3.2. We deal with updates, e.g. inserting a new square into P or deleting an existing square from P, in Sect. 3.3. In Sect. 3.4 we analyze the total number of events that we have to process, and the time required to do so, when we grow the squares.

3.1 The Data Structure

Our data structure consists of two three-layered trees T^L and T^R, and a set of certificates linking nodes from T^L and T^R. These trees essentially form two 3D range trees on the centers of the squares in P, taking the third coordinate p_γ of each point to be their rank in the order (from left to right) along the line γ. The third layer of T^L doubles as a kinetic tournament tracking the bottom left vertices of squares. Similarly, T^R tracks the top right vertices of the squares.

The Layered Trees. The tree T^L is a 3D-range tree storing the center points in P. Each layer is implemented by a BB[α] tree [11], and each node μ corresponds to a canonical subset P_μ of points stored in the leaves of the subtree rooted at μ. The points are ordered on x-coordinate first, then on y-coordinate, and finally on γ-coordinate. Let L_μ denote the set of bottom left vertices of squares corresponding to the set P_μ, for some node μ.

Consider the associated structure X_v^L of some secondary node v. We consider X_v^L as a kinetic tournament on the x-coordinates of the points L_v [1]. More specifically, every internal node $w \in X_v^L$ corresponds to a set of points P_w consecutive along the line γ. Since the γ-coordinates of a point p and its bottom left vertex ℓ_p are equal, this means w also corresponds to a set of consecutive bottom left vertices L_w. Node w stores the vertex ℓ_p in L_w with minimum x-coordinate, and will maintain certificates that guarantee this [1].

The tree T^R has the same structure as T^L: it is a three-layered range tree on the center points in P. The difference is that a ternary structure X_v^R, for some secondary node v, forms a kinetic tournament maintaining the maximum x-coordinate of the points in R_v, where R_v are the top right vertices of the squares (with center points) in P_v. Hence, every ternary node $z \in X_v^R$ stores the vertex r_q with maximum x-coordinate among R_v. Let $\mathcal{X}^L$ and $\mathcal{X}^R$ denote the set of all kinetic tournament nodes in T^L and T^R, respectively.

Linking the Trees. Next, we describe how to add *linking certificates* between the kinetic tournament nodes in the trees T^L and T^R that guarantee the squares are disjoint. More specifically, we describe the certificates, between nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$, that guarantee that the squares $\square_p$ and $\square_q$ are disjoint, for all pairs $q \in P$ and $p \in D^+(q)$.

Consider a point q. There are $O(\log^2 n)$ nodes in the secondary trees of T^L, whose canonical subsets together represent exactly $D(q)$. For each of these nodes v we can then find $O(\log n)$ nodes in X_v^L representing the points in $L^+(q)$. So, in total q is *interested in* a set $Q^L(q)$ of $O(\log^3 n)$ kinetic tournament nodes. It now follows from Lemma 3 that if we were to add certificates certifying that r_q is left of the point stored at the nodes in $Q^L(q)$ we can detect when $\square_q$ intersects with a square of a point in $D^+(q)$. However, as there may be many points q interested in a particular kinetic tournament node w, we cannot afford to maintain all of these certificates. The main idea is to represent all of these points q by a number of canonical subsets of nodes in T^R, and add certificates to only these nodes.

Consider a point p. Symmetric to the above construction, there are $O(\log^3 n)$ nodes in kinetic tournaments associated with T^R that together exactly represent the (top right corners of) the points q dominated by p and for which $p \in D^+(q)$. Let $Q^R(p)$ denote this set of kinetic tournament nodes.

Next, we extend the definitions of Q^L and Q^R to kinetic tournament nodes. To this end, we first associate each kinetic tournament node with a (query) point in $\mathbb{R}^3$. Consider a kinetic tournament node w in a tournament X_v^L, and let u be the node in the primary T^L for which $v \in T_u$. Let $\underline{m}^w = (\min_{a \in P_u} a_x, \min_{b \in P_v} b_y, \min_{c \in P_w} c_\gamma)$ be the point associated with w (note that we take the minimum over different sets P_u, P_v, and P_w for the different coordinates), and define $Q^R(w) = Q^R(\underline{m}^w)$. Symmetrically, for a node z in a tournament X_v^R, with $v \in T_u$ and $u \in T^R$, we define $\overline{m}^z = (\max_{a \in P_u} a_x, \max_{b \in P_v} b_y, \max_{c \in P_z} c_\gamma)$ and $Q^L(z) = Q^L(\overline{m}^z)$. See Fig. 3.

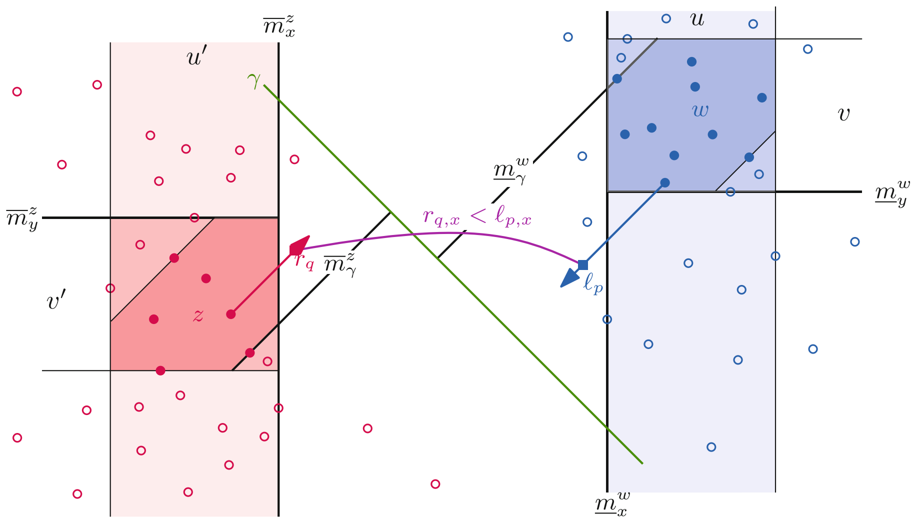


Fig. 3. The points $\overline{m}^z$ and $\underline{m}^w$ are defined by a pair of nodes $z \in \mathcal{X}_{v'}^R$, with $v' \in T_{u'}$, and $w \in X_v^L$, with $v \in T_u$. If $w \in Q^L(\overline{m}^z)$ and $z \in Q(\underline{m}^w)$ then we add a linking certificate between the rightmost upper right-vertex $r_q, q \in P_z$, and the leftmost bottom left vertex $\ell_p, p \in P_w$.

We now add a linking certificate between every pair of nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$ for which (i) w is a node in the canonical subset of z, that is $w \in Q^L(z)$, *and* (ii) vice versa, $z \in Q^R(w)$. Such a certificate will guarantee that the point r_q currently stored at z lies left of the point ℓ_p stored at w.

Lemma 4. *Every kinetic tournament node is involved in $O(\log^3 n)$ linking certificates, and thus every point p is associated with at most $O(\log^6 n)$ certificates.*

We now argue that we can still detect the first upcoming intersection.

Lemma 5. *Consider two sets of elements, say blue elements B and red elements R, stored in the leaves of two binary search trees T^B and T^R, respectively, and let $p \in B$ and $q \in R$, with $q < p$, be leaves in trees T^B and T^R, respectively. There is a pair of nodes $b \in T^B$ and $r \in T^R$, such that (i) $p \in P_b$ and $b \in C(T^B, [x', \infty))$, and (ii) $q \in P_r$ and $r \in C(T^R, (-\infty, x])$, where $x' = \max P_r, x = \min P_b$, and $C(T^S, I)$ denotes the minimal set of nodes in T^S whose canonical subsets together represent exactly the elements of $S \cap I$.*

Proof. Let b be the first node on the path from the root of T^B to p such that the canonical subset P_b of b is contained in the interval $[q, \infty)$, but the canonical subset of the parent of b is not. We define b to be the root of T^B if no such node exists. We define r to be the first node on the path from the root of T^R to q for which P_r is contained in $(-\infty, x]$ but the canonical subset of the parent is not. We again define r as the root of T^R if no such node exists (see Fig. 4). Clearly, we now have that r is one of the nodes whose canonical subsets form $R \cap (-\infty, x]$, and that $q \in P_r$ (as r lies on the search path to q). It is also easy to see that $p \in P_b$, as b lies on the search path to p. All that remains is to show that b is one of the canonical subsets that together form $B \cap [x', \infty)$. This follows from the fact that $q \leq x' < x \leq p$—and thus P_b is indeed a subset of $[x', \infty)$—and the fact that the subset of the parent v of b contains an element smaller than q, and can thus not be a subset of $[x', \infty)$. □

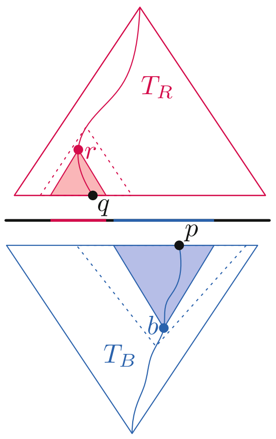


Fig. 4. The nodes b and r in the trees T^B and T^R. (Color figure online)

Lemma 6. *Let $\square_p$ and $\square_q$, with $p \in D^+(q)$, be the first pair of squares to intersect, at some time t^*, then there is a pair of nodes w, z that have a linking certificate that fails at time t^*.*

Proof. Consider the leaves representing p and q in T^L and T^R, respectively. By Lemma 5 we get that there is a pair of nodes $u \in T^L$ and $u' \in T^R$ that, among other properties, have $p \in P_u$ and $q \in P_{u'}$. Hence, we can apply Lemma 5 again on the associated trees of u and u', giving us nodes $v \in T_u$ and $v' \in T_{u'}$ which again have $p \in P_v$ and $q \in P_{v'}$. Applying Lemma 5 once more on X_v^L and $X_{v'}^R$ gives us nodes $w \in X_v^L$ and $z \in X_{v'}^R$ with $p \in P_w$ and $q \in P_z$. In addition, these three applications of Lemma 5 give us two points (x, y, γ) and (x', y', γ') where:

- P_u occurs as a canonical subset representing $P \cap ([x', \infty) \times \mathbb{R}^2)$,
- P_v occurs as a canonical subset representing $P_u \cap (\mathbb{R} \times [y', \infty) \times \mathbb{R})$, and
- P_w occurs as a canonical subset representing $P_v \cap (\mathbb{R}^2 \times [\gamma', \infty))$,

and such that

- $P_{u'}$ occurs as a canonical subset representing $P \cap ((-\infty, x] \times \mathbb{R}^2)$,
- $P_{v'}$ occurs as a canonical subset representing $P_{u'} \cap (\mathbb{R} \times (-\infty, y] \times \mathbb{R})$, and
- P_z occurs as a canonical subset representing $P_{v'} \cap (\mathbb{R}^2 \times (-\infty, \gamma])$.

Combining these first three facts, and observing that $\overline{m}^z = (x', y', \gamma')$ gives us that P_w occurs as a canonical subset representing $P \cap ([x', \infty) \times [y', \infty) \times [\gamma', \infty)) = D^+((x', y', \gamma'))$, and hence $w \in Q^L(\overline{m}^z) = Q^L(z)$. Analogously, combining the latter three facts and $\underline{m}^w = (x, y, \gamma)$ gives us $z \in Q^R(w)$. Therefore, w and z have a linking certificate. This linking certificate involves the leftmost bottom left vertex ℓ_a for some point $a \in P_w$ and the rightmost top right vertex

r_b for some point $b \in P_z$. Since $p \in P_w$ and $q \in P_z$, we have that $r_q \leq r_b$ and $\ell_a \leq \ell_p$, and thus we detect their intersection at time t^*. $\square$

From Lemma 6 it follows that we can now detect the first intersection between a pair of squares $\square_p, \square_q$, with $p \in D^+(q)$. We define an analogous data structure for when $p \in D^-(q)$. Following Lemma 3, the kinetic tournaments will maintain the vertices with minimum and maximum y-coordinate for this case. We then again link up the kinetic tournament nodes in the two trees appropriately.

Space Usage. Our trees T^L and T^R are range trees in $\mathbb{R}^3$, and thus use $O(n \log^2 n)$ space. However, it is easy to see that this is dominated by the space required to store the certificates. For all $O(n \log^2 n)$ kinetic tournament nodes we store at most $O(\log^3 n)$ certificates (Lemma 4), and thus the total space used by our data structure is $O(n \log^5 n)$. In Sect. 4 we will show that the number of certificates that we maintain is actually only $O(n(\log n \log\log n)^2)$. This means that our data structure also uses only $O(n(\log n \log\log n)^2)$ space.

3.2 Answering Queries

The basic query that our data structure supports is testing if a query square $\square_q$ currently intersects with a square $\square_p$ in P, with $p \in D^+(q)$. To this end, we simply select the $O(\log^3 n)$ kinetic tournament nodes whose canonical subsets together represent $D^+(q)$. For each node w we check if the x-coordinate of the lower-left vertex ℓ_p stored at that node (which has minimum x-coordinate among L_w) is smaller than the x-coordinate of r_q. If so, the squares intersect. The correctness of our query algorithm directly follows from Observation 2. The total time required for a query is $O(\log^3 n)$. Similarly, we can test if a given query point q is contained in a square $\square_p$, with $p \in D^+(q)$. Our complete data structure contains additional trees analogous to T^L that can be used to check if there is a square $\square_p \in P$ that intersects $\square_q$, with $p \in D^-(q)$ or p in one of the other quadrants defined by q.

3.3 Inserting or Deleting a Square

At an insertion or deletion of a square $\square_p$ we proceed in three steps. (1) We update T^L and T^R, restoring range tree properties, and ensure that the ternary data structures are correct kinetic tournaments. (2) For each kinetic tournament node in $\mathcal{X}^L$ affected by the update, we query T^R to find a new set of linking certificates. We update $\mathcal{X}^R$ analogously. (3) We update the global event queue.

Lemma 7. *Inserting or deleting a square in T^L takes $O(\log^3 n)$ amortized time.*

Proof. We use the following standard procedure for updating the three-level BB[α] trees T^L in $O(\log^3 n)$ amortized time. An update (insertion or deletion) in a ternary data structure can easily be handled in $O(\log n)$ time. When we insert into or delete an element x in a BB[α] tree that has associated data structures, we add or remove the leaf that contains x, rebalance the tree by rotations, and finally

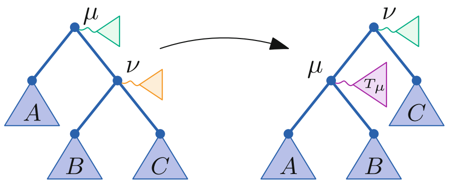


Fig. 5. After a left rotation around an edge (μ, ν), the associated data structure T_μ of node μ (pink) has to be rebuilt from scratch as its canonical subset has changed. For node ν we can reuse the old associated data of node μ. No other nodes are affected. (Color figure online)

add or remove x from the associated data structures. When we do a left rotation around an edge (μ, ν) we have to build a new associated data structure for node μ from scratch. See Fig. 5. Right rotations are handled analogously. It is well known that if building the associated data structure at node μ takes $O(|P_\mu| \log^c |P_\mu|)$ time, for some $c \geq 0$, then the costs of all rebalancing operations in a sequence of m insertions and deletions takes a total of $O(m \log^{c+1} n)$ time, where n is the maximum size of the tree at any time [10]. We can build a new kinetic tournament X_v^L for node v (using the associated data structures at its children) in linear time. Note that this cost excludes updating the global event queue. Building a new secondary tree T_v, including its associated kinetic tournaments, takes $O(|T_v| \log |T_v|)$ time. It then follows that the cost of our rebalancing operations is at most $O(m \log^2 n)$. This is dominated by the total number of nodes created and deleted, $O(m \log^3 n)$, during these operations. Hence, we can insert or delete a point (square) in T^L in $O(\log^3 n)$ amortized time. $\square$

Clearly we can update T^R in $O(\log^3 n)$ amortized time as well. Next, we update the linking certificates. We say that a kinetic tournament node w in T^L is *affected by* an update if (i) the update added or removed a leaf node in the subtree rooted at w, (ii) node w was involved in a tree rotation, or (iii) w occurs in a newly built associated tree X_v^L (for some node v). Let $\mathcal{X}_i^L$ denote the set of nodes affected by update i ($\mathcal{X}_i^R$ of T^R is defined analogously). For each node $w \in \mathcal{X}_i^L$, we query T^R to find the set of $O(\log^3 n)$ nodes whose canonical subsets represent $Q^R(w)$. For each node z in this set, we test if we have to add a linking certificate between w and z. As we show next, this takes constant time for each node z, and thus $O(\sum_i |\mathcal{X}_i^L| \log^3 n)$ time in total, for all nodes w (analogously for $\mathcal{X}_i^R$).

We have to add a link between a node $z \in Q^R(w)$ and w if and only if we also have $w \in Q^L(z)$. We test this as follows. Let v be the node whose associated tree X_v^L contains w, and let u be the node in T^L whose associated tree contains v. We have that $w \in Q^L(z)$ if and only if $u \in C(T^L, [\overline{m}_x^z, \infty))$, $v \in C(T_u, [\overline{m}_y^z, \infty))$, and $w \in C(X_v^L, [\overline{m}_\gamma^z, \infty))$. We can test each of these conditions in constant time:

Observation 8. Let q be a query point in $\mathbb{R}^1$, let w be a node in a binary search tree T, and let $x_p = \min P_p$ of the parent p of w in T, or $x_p = -\infty$ if no such node exists. We have that $w \in C(T, [q, \infty))$ if and only if $q \leq \min P_w$ and $q > x_p$.

Finally, we delete all certificates involving no longer existing nodes from our global event queue, and replace them by all newly created certificates. This takes $O(\log n)$ time per certificate. We charge the cost of deleting a certificate to when it gets created. Since every node w affected creates at most $O(\log^3 n)$ new certificates, all that remains is to bound the total number of affected nodes. Here we can use basically the same argument as when bounding the update time.

Lemma 9. *Inserting a disjoint square into P, or deleting a square from P takes $O(\log^7 n)$ amortized time.*

3.4 Running the Simulation

All that remains is to analyze the number of events processed, and the time to do so. Since each failure of a linking certificate produces an intersection, and thus an update the number of such events is at most the number of updates. To bound the number of events created by the tournament trees we use an argument similar to that of Agarwal et al. [1].

Theorem 10. *We can maintain a set P of n disjoint growing squares in a fully dynamic data structure such that we can detect the first time that a square $\square_q$ intersects with a square $\square_p$, with $p \in D^+(q)$. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time. For a sequence of m operations, the structure processes a total of $O(m\alpha(n)\log^3 n)$ events in a total of $O(m\alpha(n)\log^7 n)$ time.*

Proof. We argued the bounds on the space usage, the query time, and the update time before. All that remains is to bound the number of events processed, and the time it takes to do so.

We start by the observation that each failure of a linking certificate produces an intersection, and thus a subsequent update. It thus follows that the number of such events is at most m.

To bound the number of events created by the tournament trees we extend the argument of Agarwal et al. [1]. For any kinetic tournament node w in T^L, the minimum x-coordinate corresponds to a lower envelope of line-segments in the t, x-space. This envelope has complexity $O(|P_w^*|\alpha(|P_w^*|)) = O(|P_w^*|\alpha(n))$, where P_w^* is the multiset of points that ever occur in P_w, i.e. that are stored in a leaf of the subtree rooted at w at some time t. Hence, the number of tournament events involving node w is also at most $O(|P_w^*|\alpha(n))$. It then follows that the total number of events is proportional to the size of these sets P_w^*, over all w in our tree. As in Lemma 7, every update directly contributes one point to $O(\log^3 n)$ nodes. The remaining contribution is due to rebalancing operations, and this cost is again bounded by $O(m \log^2 n)$. Thus, the total number of events processed is $O(m\alpha(n)\log^3 n)$.

At every event, we have to update the $O(\log^3 n)$ linking certificates of w. This can be done in $O(\log^4 n)$ time (including the time to update the global event queue). Thus, the total time for processing all kinetic tournament events in T^L is $O(m\alpha(n)\log^7 n)$. The analysis for the kinetic tournament nodes z in T^R is analogous. $\square$

To simulate the process of growing the squares in P, we now maintain eight copies of the data structure from Theorem 10: two data structures for each quadrant (one for D^+, the other for D^-). We thus obtain the following result.

Theorem 11. *We can maintain a set P of n disjoint growing squares in a fully dynamic data structure such that we can detect the first time that two squares in P intersect. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time. For a sequence of m operations, the structure processes $O(m\alpha(n)\log^3 n)$ events in a total of $O(m\alpha(n)\log^7 n)$ time.*

And thus we obtain the following agglomerative glyph clustering solution.

Theorem 12. *Given a set of n initial square glyphs P, we can compute an agglomerative clustering of the squares in P in $O(n\alpha(n)\log^7 n)$ time using $O(n(\log n \log\log n)^2)$ space.*

4 Efficient Representation of Dominance Relations

The linking certificates of our data structure actually comprise an efficient representation of all dominance relations between two point sets. This representation, and in particular the tighter analysis in this section, is of independent interest.

Let R and B be two point sets in $\mathbb{R}^d$ with $|R| = n$ and $|B| = m$, and let T^R and T^B be range trees built on R and B, respectively. We assume that each layer of T^R and T^B is a BB[α]-tree. By definition, every node u on the lowest layer of T^R or T^B has an associated d-dimensional range Q_u (the hyper-box, not the subset of points). For a node $u \in T^R$, we consider the subset of points in B that dominate all points in Q_u, which can be comprised of $O(\log^d m)$ canonical subsets of B, represented by nodes in T^B. Similarly, for a node $v \in T^B$, we consider the subset of points in R that are dominated by all points in Q_v, which can be represented by $O(\log^d n)$ nodes in T^R. We now link a node $u \in T^R$ and a node $v \in T^B$ if and only if v represents such a canonical subset for u and vice versa. By repeatedly applying Lemma 5 for each dimension, it can easily be shown that these links represent all dominance relations between R and B.

As a d-dimensional range tree consists of $O(n \log^{d-1} n)$ nodes, a trivial bound on the number of links is $O(m \log^{2d-1} n)$ (assuming $n \geq m$). Below we show that the number of links can be bounded by $O(n(\log n \log\log n)^{d-1})$.

Analyzing the Number of Links in 1D. Let R and B be point sets in $\mathbb{R}$ with $|R| = n, |B| = m$, and $n \geq m$. Now, every associated range of a node u in T^R or T^B is an interval I_u. We extend the interval to infinity in one direction;

to the left for $u \in T^R$, and to the right for $u \in T^B$. For analysis purposes we construct another range tree T on $R \cup B$, where T is not a BB[α]-tree, but instead a perfectly balanced tree with height $\lceil \log(n+m) \rceil$. For convenience we slightly expand the associated intervals of T so that all points in $R \cup B$ are interior to the associated intervals. We associate a node u in T^R or T^B with a node v in T if the endpoint of I_u is contained in the associated interval I_v of v.

Observation 13. Nodes of T^R or T^B are associated with at most one node per level of T.

For two intervals $I_u = (-\infty, a]$ and $I_v = [b, \infty)$, corresponding to a node $u \in T^R$ and a node $v \in T^B$, let $[a, b]$ be the *spanning interval* of u and v. We now want to charge spanning intervals of links to nodes of T. We charge a spanning interval $I_{uv} = [a, b]$ to a node w of T if and only if $[a, b]$ is a subset of I_w, and $[a, b]$ is cut by the splitting coordinate of w. Clearly, every spanning interval can be charged to exactly one node of T. Now, for a node u of T, let $h_R(u)$ be the height of the highest node of T^R associated with u, and let $h_B(u)$ be the height of the highest node of T^B associated with u.

Lemma 14. *$O(h_R(u) \cdot h_B(u))$ spanning intervals are charged to a node u of T.*

Proof. Let x be the splitting coordinate of u and let $r \in T^R$ and $b \in T^B$ form a spanning interval that is charged to u. We claim that, using the notation introduced in Lemma 5, $r \in C(T^R, (-\infty, x])$ (and symmetrically, $b \in C(T^B, [x, \infty))$). Let $I_b = [x', \infty)$ be the associated interval of b, where $x' > x$. By definition, $r \in C(T^R, (-\infty, x'])$. If $r \notin C(T^R, (-\infty, x])$, then the right endpoint of I_r must lie between x and x'. But then the spanning interval of r and b would not be charged to u. As a result, we can only charge spanning intervals between $h_R(u)$ nodes of T^R and $h_B(u)$ nodes of T^B, of which there are $O(h_R(u) \cdot h_B(u))$. □

Using Lemma 14, we count the total number of charged spanning intervals and hence, links between T^R and T^B. We refer to this number as $N(T^R, T^B)$. This is simply $\sum_{u \in T} O(h_R(u) \cdot h_B(u)) \le \sum_{u \in T} O(h_R(u)^2 + h_B(u)^2)$. We can split the sum and assume w.l.o.g. that $N(T^R, T^B) \le 2 \sum_{u \in T} O(h_R(u)^2)$. Rewriting the sum based on heights in T^R and writing $n_T(h_R)$ for the number of nodes of T that have a node of height h_R associated with it gives

$$N(T^R, T^B) \le \sum_{h_R = 0}^{\mathit{height}(T^R)} n_T(h_R) \cdot O(h_R^2).$$

To bound $n_T(h)$ we use Observation 13 and the fact that T^R is a BB[α] tree. Let $c = \frac{1}{1-\alpha}$, then we get that $\mathit{height}(T^R) \le \log_c(n)$ from properties of BB[α] trees. Therefore, the number of nodes in T^R that have height h is at most $O(\frac{n}{c^h})$.

Lemma 15. $n_T(h) = O\left(\frac{(n+m)h}{c^h}\right)$.

Proof. As argued, there are at most $O(n/c^h)$ nodes in T^R of height h. Consider cutting the tree T at level $\log(n/c^h)$. This results in a top tree of size $O(n/c^h)$, and $O(n/c^h)$ bottom trees. Clearly, the top tree contributes at most its size to $n_T(h)$. All bottom trees have height at most $\lceil \log(n+m) \rceil - \log(n/c^h) = O(\log(c^h)+\log(1+m/n)) = O(h+m/n)$. Every node in T^R of height h can, in the worst case, be associated with one distinct node per level in the bottom trees by Observation 13. Hence, the bottom trees contribute at most $O(n(h+m/n)/c^h) = O((nh+m)/c^h) = O((n+m)h/c^h)$ to $n_T(h)$. □

Using this bound on $n_T(h)$ in the sum we previously obtained gives

$$N(T^R, T^B) \leq \sum_{h_R=0}^{height(T^R)} O\left(\frac{(n+m)h_R^3}{c^{h_R}}\right) \leq O(n+m) \sum_{h=0}^{\infty} \frac{h^3}{c^h} = O(n+m).$$

Where indeed, $\sum_{h=0}^{\infty} \frac{h^3}{c^h} = O(1)$ because $c > 1$. Thus, we conclude:

Theorem 16. *The number of links between two 1-dimensional range trees T^R and T^B containing n and m points, respectively, is bounded by $O(n+m)$.*

Extending to Higher Dimensions. We now extend the bound to d dimensions. We first determine the links for the top-layer of the range trees. This results in links between associated range trees of $d-1$ dimensions (see Fig. 6). We then bound the number of links within the linked associated trees by induction on d.

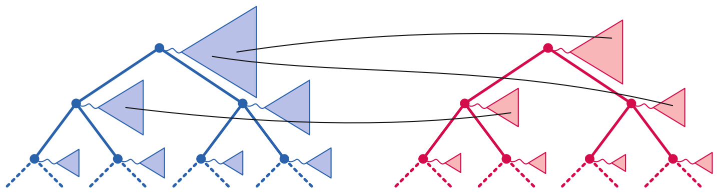


Fig. 6. Two layered trees with two layers, and the links between them (sketched in black). We are interested in bounding the number of such links.

Theorem 17. *The number of links between two d-dimensional range trees T^R (on n points) and T^B (on $m \leq n$ points), is bounded by $O(n(\log n \log\log n)^{d-1})$.*

Proof. We show by induction on d that the number of links is bounded by the minimum of $O(n(\log n \log\log n)^{d-1})$ and $O(m \log^{2d-1} n)$. The second bound is the trivial bound stated above. The base case $d = 1$ is provided by Theorem 16. Consider the case $d > 1$. We first determine the links for the top-layer of T^R and T^B. Now consider the links between an associated tree T_u in T^R containing k points and other associated trees $T_0, \ldots, T_r$ that contain at most k points. Since T_u can be linked with only one associated tree per level, and because both range trees use BB[α] trees, the number of points $m_0, \ldots, m_r$ in $T_0, \ldots, T_r$

satisfy $m_i \leq k/c^i$ $(0 \leq i \leq r)$ where $c = \frac{1}{1-\alpha}$. By induction, the number of links between T_u and T_i is bounded by the minimum of $O(k(\log n \log\log n)^{d-2})$ and $O(m_i \log^{2d-3} n)$. Now let $i^* = \log_c(\log^{d-1} n) = O(\log\log n)$. Then, for $i \geq i^*$, we get that $O(m_i \log^{2d-3} n) = O(k \log^{d-2} n)$. Since the sizes of the associated trees decrease geometrically, the total number of links between T_u and T_i for $i \geq i^*$ is bounded by $O(k \log^{d-2} n)$. The links with the remaining trees can be bounded by $O(k \log^{d-2} n(\log\log n)^{d-1})$. Finally note that the top-layer of each range tree has $O(\log n)$ levels, and that each level contains n points in total. Thus, we obtain $O(n \log^{d-1} n(\log\log n)^{d-1})$ links in total. The remaining links for which the associated tree in T^B is larger than in T^R can be bounded analogously. □

It follows from Theorem 17 that the number of certificates maintained, and thus the space used, by our data structure from Sect. 3 is only $O(n(\log n \log\log n)^2)$.

References

1. Agarwal, P.K., Kaplan, H., Sharir, M.: Kinetic and dynamic data structures for closest pair and all nearest neighbors. ACM Trans. Algorithms **5**(1), 4:1–4:37 (2008)
2. Ahn, H.-K., Bae, S.W., Choi, J., Korman, M., Mulzer, W., Oh, E., Park, J.-W., van Renssen, A., Vigneron, A.: Faster algorithms for growing prioritized disks and rectangles. In: International Symposium on Symbolic and Algebraic Computation, pp. 1–13 (2017)
3. Alexandron, G., Kaplan, H., Sharir, M.: Kinetic and dynamic data structures for convex hulls and upper envelopes. Comput. Geom. Theory Appl. **36**(2), 144–1158 (2007)
4. Bahrdt, D., Becher, M., Funke, S., Krumpe, F., Nusser, A., Seybold, M., Storandt, S.: Growing balls in $\mathbb{R}^d$. In: Proceedings of the 19th Workshop on Algorithm Engineering and Experiments, pp. 247–258 (2017)
5. Castermans, T., Speckmann, B., Staals, F., Verbeek, K.: Agglomerative clustering of growing squares. ArXiv e-prints (2017)
6. Castermans, T., Speckmann, B., Verbeek, K., Westenberg, M.A., Betti, A., van den Berg, H.: GlamMap: geovisualization for e-humanities. In: Proceedings of the 1st Workshop on Visualization for the Digital Humanities (2016)
7. Funke, S., Krumpe, F., Storandt, S.: Crushing disks efficiently. In: Mäkinen, V., Puglisi, S.J., Salmela, L. (eds.) IWOCA 2016. LNCS, vol. 9843, pp. 43–54. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44543-4_4
8. Funke, S., Storandt, S.: Parametrized runtimes for ball tournaments. In: Proceedings of the 33rd European Workshop on Computational Geometry, pp. 221–224 (2017)
9. Guibas, L.: Kinetic data structures. In: Mehta, D.P., Sahni, S. (eds.) Handbook of Data Structures and Applications, pp. 23:1–23:18. CRC Press, Boca Raton (2004)
10. Mehlhorn, K.: Data Structures and Algorithms 1: Sorting and Searching. Springer, Heidelberg (1984). https://doi.org/10.1007/978-3-642-69672-5
11. Nievergelt, J., Reingold, E.M.: Binary search trees of bounded balance. SIAM J. Comput. **2**(1), 33–43 (1973)

Fourier Entropy-Influence Conjecture for Random Linear Threshold Functions

Sourav Chakraborty[1,2], Sushrut Karmalkar[3], Srijita Kundu[4], Satyanarayana V. Lokam[5], and Nitin Saurabh[6](✉)

[1] Chennai Mathematical Institute, Chennai, India
sourav@cmi.ac.in
[2] Centrum Wiskunde Informatika, Amsterdam, Netherlands
[3] University of Texas, Austin, USA
sushrutk@cs.utexas.edu
[4] Centre for Quantum Technologies, Singapore, Singapore
srijita.kundu@u.nus.edu
[5] Microsoft Research, Bangalore, India
Satya.Lokam@microsoft.com
[6] Charles University, Prague, Czech Republic
nitin@iuuk.mff.cuni.cz

Abstract. The Fourier-Entropy Influence (FEI) Conjecture states that for any Boolean function $f : \{+1, -1\}^n \to \{+1, -1\}$, the Fourier entropy of f is at most its influence up to a universal constant factor. While the FEI conjecture has been proved for many classes of Boolean functions, it is still not known whether it holds for the class of Linear Threshold Functions. A natural question is: Does the FEI conjecture hold for a "random" linear threshold function? In this paper, we answer this question in the affirmative. We consider two natural distributions on the weights defining a linear threshold function, namely uniform distribution on $[-1, 1]$ and Normal distribution.

1 Introduction

Boolean functions are the most basic object of study in Theoretical Computer Science. There are many complexity measures associated with a Boolean function $f\colon \{+1, -1\}^n \to \{+1, -1\}$, e.g., *degree*, *sensitivity*, *certificate complexity*, etc. These measures provide lower bound on the complexity of a Boolean function in different models of computation (e.g., PRAM, Circuits/Formulas, Decision trees, etc.).

A particularly interesting one is the *average sensitivity* of a Boolean function. Given $f\colon \{+1, -1\}^n \to \{+1, -1\}$, and an input $x \in \{+1, -1\}^n$, sensitivity of f at x, denoted $\mathsf{s}(f, x)$, is defined as the number of neighbours of x in the Hamming cube where f takes different value than at x. The *average sensitivity* of f, denoted

N. Saurabh—The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 616787.

$\mathsf{as}(f)$, is the average of $\mathsf{s}(f,x)$, $\mathbb{E}_x[\mathsf{s}(f,x)]$, under the uniform distribution. In the definition above, we fixed an $x \in \{+1,-1\}^n$ and looked at the variables the value of $f(x)$ depends on. Similarly, we can fix a variable $i \in [n]$, and look at the number of inputs x such that the value $f(x)$ depends on this variable. This leads us to the notion of *influence*. The *influence* of the i-th variable, denoted $\mathsf{Inf}_i(f)$, is defined to be $\Pr_x[f(x) \neq f(x^i)]$, where x is chosen uniformly at random and x^i denotes x with i-th variable negated. The *influence* of f, $\mathsf{Inf}(f)$, is defined as $\sum_{i=1}^n \mathsf{Inf}_i(f)$. It is easily seen that $\mathsf{Inf}(f)$ equals $\mathsf{as}(f)$. Influence is related to many complexity measures, and many of these relations can be established via Fourier analysis. For example, circuit size [1,2], formula size [3], decision trees [4], etc. For an in-depth treatment of Fourier analysis in Boolean functions we refer to [5].

Every Boolean function $f\colon \{+1,-1\}^n \to \{+1,-1\}$ has a unique representation in the Fourier basis :

$$f(x) = \sum_{S\subseteq[n]} \widehat{f}(S) \prod_{i\in S} x_i \enspace .$$

The set of Fourier coefficients is given by $\{\widehat{f}(S)\}_{S\subseteq[n]}$. It follows from Parseval's identity that $\sum_{S\subseteq[n]} \widehat{f}(S)^2 = \mathbb{E}_x[f(x)^2] = 1$. Thus, the squared Fourier coefficients naturally define a distribution over the subsets of $\{1,2,\dots,n\}$. The Fourier-entropy of f, denoted $\mathbb{H}(f)$, is defined to be the Shannon entropy of this distribution. That is,

$$\mathbb{H}(f) := \sum_{S\subseteq[n]} \widehat{f}(S)^2 \log \frac{1}{\widehat{f}(S)^2} \enspace .$$

A longstanding and important conjecture in Analysis of Boolean functions states that the Fourier entropy of a Boolean function is bounded above by the total influence of the function up to a constant factor. More formally,

Fourier-Entropy Influence (FEI) Conjecture: There exists a universal constant $C > 0$ such that for all $f\colon \{+1,-1\}^n \to \{+1,-1\}$, $\mathbb{H}(f) \leqslant C \cdot \mathsf{Inf}(f)$. That is,

$$\sum_{S\subseteq[n]} \widehat{f}(S)^2 \log \frac{1}{\widehat{f}(S)^2} \leqslant C \cdot \sum_{S\subseteq[n]} |S| \widehat{f}(S)^2 \enspace . \tag{1}$$

The conjecture was made by Friedgut and Kalai [6] in 1996. The genesis of the conjecture is in the study of threshold phenomena in random graphs [6]. For example, it implies a lower bound of $\Omega(\log^2 n)$ on the influence of any n-vertex monotone graph property. The current best lower bound, by Bourgain and Kalai [7], is $\Omega(\log^{2-\epsilon} n)$, for any constant $\epsilon > 0$. However, over time, many non-trivial implications have been observed. In particular, it implies a variant of *Mansour's Conjecture* [8] that is sufficient to imply a polynomial time *agnostic* learning algorithm for DNFs [9], for any constant error parameter, resolving a major open problem [10] in computational learning theory.

Mansour's Conjecture (variant): Let $f\colon \{+1,-1\}^n \to \{+1,-1\}$ be any Boolean function computable by a t-term DNF formula. Then, for any $\epsilon > 0$, there exists a polynomial p with $t^{O(1/\epsilon)}$ terms such that $\mathbb{E}[(f-p)^2] \leqslant \epsilon$. (Mansour conjectured the exponent of t to be $O(\log \frac{1}{\epsilon})$ [8].)

In general, the FEI conjecture implies *sparse L_2-approximations* for Boolean functions. That is, it implies the existence of a polynomial p with $2^{O(\mathsf{Inf}(f)/\epsilon)}$ terms such that $\mathbb{E}[(f-p)^2] \leqslant \epsilon$, for any $\epsilon > 0$. Presently, the best known construction [11] yields a bound of $2^{O((\mathsf{Inf}(f)/\epsilon)^2)}$ on the number of terms.

Finally, the FEI inequality (1) is also known to imply the famous Kahn-Kalai-Linial theorem [12].

KKL Theorem: For any Boolean function $f\colon \{+1,-1\}^n \to \{+1,-1\}$, there exists an $i \in [n]$ such that $\mathsf{Inf}_i(f) = \Omega\left(\mathrm{Var}(f)\frac{\log n}{n}\right)$.

In fact, as observed by [13], the following weakening of the FEI conjecture suffices to imply the KKL theorem.

Fourier Min-Entropy-Influence (FMEI) Conjecture: There exists a universal constant $C > 0$ such that for all $f\colon \{+1,-1\}^n \to \{+1,-1\}$,

$$\min_{S\subseteq[n]} \log \frac{1}{\widehat{f}(S)^2} \leqslant C \cdot \mathsf{Inf}(f).$$

In other words, it conjectures how large the maximum Fourier coefficient must be. That is, there exists a set $S \subseteq [n]$ such that $\widehat{f}(S)^2 \geqslant 2^{-C\cdot\mathsf{Inf}(f)}$.

For more on the FEI conjecture we refer the interested reader to Gil Kalai's blog post [14].

While the FEI and the FMEI conjectures have resisted solutions for long, it is easy to verify the conjectures for simple functions such as OR, AND, Tribes, $\mathsf{Majority}$, etc. Thus, researchers have tried to establish the conjectures for special classes of Boolean functions. In particular, it has been shown that FEI conjecture holds for random DNFs [15], symmetric functions and read-once decision trees [13], random functions [16], read-once formulas [17,18], decision trees with bounded average depth [18,19], and bounded read decision trees [19]. In [13] it was also observed that FMEI conjecture holds for monotone functions.

In this paper, we study the FEI conjecture for the class of *linear threshold functions* (LTF). A Boolean function $f\colon \{+1,-1\}^n \to \{+1,-1\}$ is said to be an LTF if there exists $w_0, w_1, \dots, w_n \in \mathbb{R}$ such that for all $x \in \{+1,-1\}^n$, $f(x) = \mathsf{sign}(w_0 + w_1x_1 + \cdots + w_nx_n)$, where, the sign is the function that maps positive values to $+1$ and negative values to -1. We assume, without loss of generality, $\mathsf{sign}(0) = -1$.

We observe that the KKL theorem implies Fourier Min-Entropy-Influence conjecture holds for linear threshold functions using the same idea from [13]. But for the case of the FEI conjecture we still cannot show that it holds for linear threshold functions in general. So a natural question to ask is:

Does FEI conjecture hold for a random LTF?

Similar question has been asked and answered for other classes of functions (for random DNFs [15] and for random Boolean functions [16]). A natural way of defining a random LTF is when the coefficients $w_0, \dots, w_n$ are drawn from a distribution $\mathcal{D}$. Choosing $w_0, \dots, w_n$ is equivalent to choosing a direction in $(n+1)$-dimensions. Thus, a natural way of sampling an LTF is to sample a unit vector in $(n+1)$-dimensions. There are many ways to sample a unit vector uniformly [20–23]. It is well known that a simple way to sample a unit vector uniformly in $(n+1)$-dimensions is to sample each coordinate from standard normal distribution and then normalize the vector. Specifically, we consider for $\mathcal{D}$ the standard normal distribution and the uniform distribution over $[-1, 1]$. To establish our main result, Theorem 1, we prove a generic technical result that says that the FEI conjecture holds with very high probability as long as $\mathcal{D}$ possesses some "nice" properties. Informally, these properties say that a centered random variable with variance 1 and bounded third absolute moment has at least a constant probability mass above the third absolute moment. Our main results are stated below for two specific natural distributions.

Theorem 1 (Main) *[Informal]*

1. *If $w_0, \dots, w_n$ are drawn from the normal distribution $N(0, 1)$ and $f(x) = \mathsf{sign}(w_0 + w_1x_1 + \cdots + w_nx_n)$ then with high probability the FEI Conjecture holds for f.*
2. *If $w_0, \dots, w_n$ are drawn from the uniform distribution over $[-1, 1]$ and $f(x) = \mathsf{sign}(w_0 + w_1x_1 + \cdots + w_nx_n)$ then with high probability the FEI Conjecture holds for f.*

We also identify (Corollary 10) certain subclasses of linear threshold functions for which FEI conjecture holds.

1.1 Our Proof Technique

To prove the FEI conjecture for a function f (or, any class of Boolean functions) one needs to provide an upper bound on the Fourier entropy of f and a matching (up to a constant factor) lower bound on the influence of f. For upper bounding the Fourier entropy of a LTF we crucially use the following theorem proved in [18].

Theorem 2 [18]**.** *If $f : \{+1, -1\}^n \to \{+1, -1\}$ is* any *linear threshold function, then $\mathbb{H}(f) \leqslant C \cdot \sqrt{n}$, where, C is a universal constant.*

Thus if f is an LTF with influence lower bounded by $\Omega(\sqrt{n})$, then using Theorem 2, we conclude that FEI conjecture holds for f. Our main contribution in the paper is to prove that a "random" LTF has influence $\Omega(\sqrt{n})$.

We provide two techniques of proving a lower bound on the influence of an LTF f. The first technique (Theorem 4) is a simple application of the Khintchine Inequality [24]. While this lower bound technique is easy and simple, this does not yield a high probability statement about the event $\mathsf{Inf}(f) = \Omega(\sqrt{n})$ when the coefficients are distributed according to the normal distribution (see Sect. 3.2). For this, we need the second technique.

The starting point in this case is the following basic inequality for LTF's.

$$\mathsf{Inf}_i(f) = \Pr_{x\in\{+1,-1\}^n}\left[-|w_i| < w_0 + \sum_{j\leqslant n:\, j\neq i} w_j x_j \leqslant |w_i|\right] . \tag{2}$$

We bound this probability using concentration inequalities. In particular, we crucially use an optimal version of the Berry-Esseen Theorem (Theorem 7) that was proved recently by Shevtsova [25]. Using the Berry-Esseen Theorem we show, informally, if a variable has a "high" weight then it has "high" influence. More precisely, we prove the following technical lemma en route to establishing the main theorem. Consider a symmetric distribution $\mathcal{D}$ around 0 with variance 1.

Lemma 3. *Let $w_j \sim \mathcal{D}$ for $1 \leqslant j \leqslant n$. For all $i \in [n]$, $\alpha \in \mathbb{R}^+$, and $\delta > 0$, with probability at least $1 - e^{-\Omega(n\mu_3^2)}$ over the choices of w_j's,*

$$\Pr_{x\in\{+1,-1\}^n}\left[\left|\sum_{1\leqslant j\leqslant n:\, j\neq i} w_j x_j\right| \leqslant \alpha\right] \geqslant \frac{\theta}{\sqrt{n}} - O\left(\frac{1}{n^{2/3}}\right),$$

where $\theta = (\alpha - \mu_3(1 + \frac{2\delta}{1-\delta}))/\sqrt{2\pi(1+\delta)}$, and $\mu_3 = \mathbb{E}_{w\sim\mathcal{D}}[|w|^3]$.

Observe that for the statement to be non-trivial $\alpha > \mu_3 + \delta'$ for some $\delta' > 0$. The optimality of Shevtsova's theorem (Theorem 7) is crucial to the fact that α can be chosen to be only slightly larger than μ_3, by an *additive* constant rather than $C \cdot \mu_3$ for some large constant $C > 1$. This difference is significant when $\mathcal{D}$ is a *truncated* distribution. A truncated distribution is a conditional distribution that results from restricting the support of some other probability distribution.

Organization of the paper: In Sect. 2 we present the two techniques to lower bound the influence of an LTF. We then show that the FEI conjecture holds for a 'random' LTF in Sect. 3, and conclude with some observations in Sect. 4.

2 Lower Bounds on Influence

Recall that $f\colon \{+1,-1\}^n \to \{+1,-1\}$ is a linear threshold function if there exists $n+1$ real numbers, $w_0, w_1, \ldots, w_n$, such that

$$\forall x = x_1 \ldots x_n \in \{+1,-1\}^n\colon f(x) = \mathsf{sign}(w_0 + w_1x_1 + \cdots + w_nx_n),$$

where $\mathsf{sign}(z) = 1$ if $z > 0$, and -1 if $z \leqslant 0$. It is clear that there are infinitely many $(n+1)$-tuples that define the same Boolean function f. Nevertheless, relationships among w_i's characterize many properties of the function. In fact, the influence of f is a function on the w_i's. Our goal is to obtain lower bounds for this function using various properties of w_i's. In this section we present two such lower bounds.

The first lower bound that we obtain is a simple observation and the lower bound is in terms of the ℓ_2-norm of the w_i's and the maximum value of the w_i's.

Theorem 4. *Let $f\colon \{+1,-1\}^n \to \{+1,-1\}$ be such that $f(x) = \mathsf{sign}(w_0 + \sum_{i=1}^n x_i w_i)$ for some weights $w_0, w_1, \ldots, w_n \in \mathbb{R}$. Then,*

$$\mathsf{Inf}(f) \geqslant \frac{\sqrt{\sum_{i=0}^n w_i^2}}{2\sqrt{2}\max_i |w_i|} - |\hat{f}(\emptyset)| \geqslant \frac{\sqrt{\sum_{i=0}^n w_i^2}}{2\sqrt{2}\max_i |w_i|} - 1.$$

The proof is omitted. It follows from an easy application of the well-known Khintchine Inequality.

Theorem 5 (Khintchine Inequality) [24]**.** *Let $w_1, w_2, \ldots, w_n \in \mathbb{R}$ be such that $\sum_i w_i^2 = 1$. Then, $\mathbb{E}_{x\in\{+1,-1\}^n}[|w_1x_1 + \cdots + w_nx_n|] \geqslant \frac{1}{\sqrt{2}}$.*

The second lower bound we obtain is more sophisticated lower bound. Recall that the i-th influence of f is given by $\Pr_{x\in\{+1,-1\}^n}[f(x) \neq f(x^i)]$. Using the fact that f is an LTF, it is easily seen to be equivalent to Eq. (2). We bound the expression in (2) from below to establish the lower bound.

Theorem 6 *Let $w_0, w_1, \ldots, w_n \in \mathbb{R}$ and let $f : \{+1,-1\}^n \to \{+1,-1\}$ be defined by $f(x) = \mathsf{sign}(w_0 + \sum_{i=1}^n x_i w_i)$. Then, for $1 \leqslant i \leqslant n$,*

$$\mathsf{Inf}_i(f) \geqslant \frac{1}{2\sqrt{2\pi(n-1)B_i}}\left(|w_i| - \frac{A_i}{B_i}\right) - \frac{|w_i|^3}{6\sqrt{2\pi}((n-1)B_i)^{3/2}} - \frac{3.4106\ A_i^{4/3}}{(n-1)^{2/3}B_i^2},$$

where $A_i := \frac{1}{n-1}\sum_{0\leqslant j\neq i\leqslant n} |w_j|^3$ and $B_i := \frac{1}{n-1}\sum_{0\leqslant j\neq i\leqslant n} |w_j|^2$.

A crucial ingredient in the proof is the following optimal version of Berry-Esseen Theorem that bounds the uniform distance between the cumulative distribution function of the standard normal distribution $N(0,1)$ and the standardized sum of independent symmetric Bernoulli random variables.

Theorem 7 (Corollary 4.19 in [25]**).** *Let $X_1, \ldots, X_n$ be independent symmetric Bernoulli random variables such that for all $1 \leqslant j \leqslant n$ there exist a_j with $\Pr[X_j = a_j] = \Pr[X_j = -a_j] = \frac{1}{2}$. If $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, then,*

$$\sup_x \left| \Pr\left[\frac{X_1 + \cdots + X_n}{\sqrt{\sum_{j=1}^n a_j^2}} < x\right] - \Phi(x)\right| \leqslant \frac{\ell_n}{\sqrt{2\pi}} + 3.4106 \cdot \ell_n^{4/3},$$

where $\ell_n = (\sum_j |a_j|^3)/(\sum_j a_j^2)^{3/2}$.

The optimality is in the fact that the constant of $1/\sqrt{2\pi}$ (as a coefficient of ℓ_n) is the best possible.

We will prove Theorem 6 in two stages. First we establish it under the assumption that $w_0 = 0$. Then, in Appendix A, we reduce the non-homogeneous case ($w_0 \neq 0$) to the homogeneous case ($w_0 = 0$) to complete the proof.

2.1 Proof of Theorem 6 (assuming $w_0 = 0$)

As mentioned earlier, the full proof of Theorem 6 will follow from the discussions in Appendix A. Thus, for some $w_1, \ldots, w_n \in \mathbb{R}$, we have $f(x) = \mathsf{sign}(\sum_{i=1}^n w_i x_i)$ for all $x \in \{+1, -1\}^n$. Relaxing the estimate in Eq. (2), we have

$$\mathsf{Inf}_i(f) \geqslant \frac{1}{2} \Pr_{x \in \{+1,-1\}^n} \left[\left| \sum_{j \leqslant n:\, j \neq i} w_j x_j \right| \leqslant |w_i| \right].$$

To obtain the claimed lower bound on influence, we further lower bound the above expression. We start with some definitions required for the proof.

For any $1 \leqslant i \leqslant n$, we define

$$A_i := \frac{1}{n-1} \sum_{j \neq i} |w_j|^3, \quad \text{and} \quad B_i := \frac{1}{n-1} \sum_{j \neq i} |w_j|^2.$$

Note that the proof of Theorem 6, for the case when $w_0 = 0$, follows by substituting $\alpha = |w_i|$ in the following lemma.

Lemma 8. *For all $i \in [n]$ and any $\alpha \in \mathbb{R}^+$,*

$$\frac{1}{2} \Pr_{x \in \{+1,-1\}^n} \left[\left| \sum_{j \leqslant n:\, j \neq i} w_j x_j \right| \leqslant \alpha \right] \geqslant \frac{1}{\sqrt{2\pi(n-1)B_i}} \left(\alpha - \frac{A_i}{B_i} \right) - \frac{\alpha^3}{6\sqrt{2\pi}((n-1)B_i)^{3/2}} - \frac{3.4106\, A_i^{4/3}}{(n-1)^{2/3} B_i^2}.$$

Proof. For any $\alpha > 0$, by the symmetry of distribution over $\sum w_j x_j$, we have

$$\begin{aligned}\frac{1}{2} \Pr_x \left[\left| \sum_{j \leqslant n:\, j \neq i} w_j x_j \right| \leqslant \alpha \right] &= \Pr_x \left[0 \leqslant \sum_{j \leqslant n:\, j \neq i} w_j x_j \leqslant \alpha \right] \\ &= \Pr_x \left[\frac{\sum_{j \leqslant n:\, j \neq i} w_j x_j}{\sqrt{\sum_{j \leqslant n:\, j \neq i} w_j^2}} \leqslant \frac{\alpha}{\sqrt{\sum_{j \leqslant n:\, j \neq i} w_j^2}} \right] - \frac{1}{2}.\end{aligned}$$

Using Theorem 7 on random variables $w_j x_j$'s, we obtain the following lower bound

$$\frac{1}{2} \Pr_x \left[\left| \sum_{j \leqslant n:\, j \neq i} w_j x_j \right| \leqslant \alpha \right] \geqslant \Phi \left(\frac{\alpha}{\sqrt{\sum_{j \leqslant n:\, j \neq i} w_j^2}} \right) - \left(\frac{1}{2} \right) - \frac{L_{(n-1)}}{\sqrt{2\pi}} - 3.4106 \cdot L_{(n-1)}^{4/3}, \tag{3}$$

where

$$L_{(n-1)} = \frac{\sum_{j \leqslant n:\, j \neq i} |w_j|^3}{(\sum_{j \leqslant n:\, j \neq i} |w_j|^2)^{3/2}} = \frac{(n-1)A_i}{((n-1)B_i)^{3/2}}.$$

Thus we have

$$-\frac{L_{(n-1)}}{\sqrt{2\pi}} - 3.4106 \cdot L_{(n-1)}^{4/3} = -\frac{1}{\sqrt{2\pi}} \cdot \frac{A_i}{\sqrt{(n-1)} B_i^{3/2}} - 3.4106 \frac{A_i^{4/3}}{(n-1)^{2/3} B_i^2}. \tag{4}$$

Also, using the following Maclaurin series

$$\Phi(x) - \frac{1}{2} = \frac{1}{\sqrt{2\pi}}\left(x - \frac{1}{6}x^3 + \frac{1}{40}x^5 - \cdots\right),$$

we have

$$\Phi\left(\frac{\alpha}{\sqrt{\sum_{j\leqslant n:\, j\neq i} w_j^2}}\right) - \frac{1}{2} \geqslant \frac{\alpha}{\sqrt{2\pi B_i(n-1)}} - \frac{\alpha^3}{6\sqrt{2\pi}((n-1)B_i)^{3/2}}. \tag{5}$$

Using Eqs. (5) and (4) in Eq. (3), we obtain

$$\frac{1}{2}\Pr_x\left[\left|\sum_{j\leqslant n:\, j\neq i} w_j x_j\right| \leqslant \alpha\right] \geqslant \frac{1}{\sqrt{2\pi(n-1)B_i}}\left(\alpha - \frac{A_i}{B_i}\right) - \frac{\alpha^3}{6\sqrt{2\pi}((n-1)B_i)^{3/2}} - \frac{3.4106\ A_i^{4/3}}{(n-1)^{2/3}B_i^2}.$$

□

In our applications of the lemma, the second and third terms will be negligible compared to the first term.

2.2 Fourier Entropy Influence Conjecture for a Class of Linear Threshold Functions

In this section we identify a class of LTFs for which we prove the FEI conjecture. The class of functions is τ-regular functions.

Definition 9. *Suppose $w_0, w_1, \ldots, w_n$ are real numbers such that $\sum_{i=0}^{n} w_i^2 = 1$. If for all $0 \leqslant i \leqslant n$, $|w_i| \leqslant \tau$ then the linear threshold function, f defined as $f(x_1, \ldots, x_n) = \mathsf{sign}(w_0 + \sum_{i=1}^{n} w_i x_i)$ is called a τ-regular LTF.*

From Theorems 2 and 4 we can show that τ-regular LTFs satisfy the FEI conjecture.

Corollary 10. *If $\tau \leqslant c/\sqrt{n}$ and if f is a τ-regular function then*

$$\mathbb{H}(f) \leqslant O(\mathsf{Inf}(f)),$$

where the constant in the Big-oh notation depends on c.

3 Lower Bounds on Influence for Random Linear Threshold Functions

Given that we still cannot prove the FEI conjecture for all Linear Threshold Functions a natural question is whether FEI conjecture holds for a "random" LTF.

Suppose $w_0, w_1, \ldots, w_n$ are drawn independently from a distribution $\mathcal{D}$. Consider the function $f(x)$ where

$$\forall x = x_1, \ldots, x_n \in \{+1, -1\}^n : f(x) = \mathsf{sign}(w_0 + w_1 x_1 + \cdots + w_n x_n).$$

Without loss of generality we can assume that the distribution is symmetric around the origin and thus the mean of the distribution $\mu(\mathcal{D})$ is 0. Also since the function remains same even if we scale the w_i's by any value, so we can also assume that the variance of the distribution $\sigma(\mathcal{D})$ is 1.

As a step towards proving the FEI conjecture for f we need to lower bound the influence of f. Note that Theorems 4 and 6 give lower bounds on the influence of f in terms of w_i's. Using similar arguments we can obtain a lower bound on the influence that holds for a random f (that is, when w_i's are drawn from a distribution $\mathcal{D}$) with high probability.

For any $n \in \mathbb{N}$, let $w_1, \ldots, w_n \sim \mathcal{D}$ be the outcome of n independent samples according to $\mathcal{D}$. We define, for any $\alpha \in \mathbb{R}^+$, $p_{\mathcal{D},n}(\alpha) = \Pr[\max_{i=1}^n\{|w_i|\} \geqslant \alpha]$.

Corollary 11 (Corollary of Theorem 4). *For any $\alpha \in \mathbb{R}^+$, with probability at least $1 - p_{\mathcal{D},n+1}(\alpha) - o(1)$,*

$$\mathsf{Inf}(f) \geqslant \Omega(\sqrt{n}/\alpha).$$

Using Bernstein's inequality it is easily seen that $\sum_i w_i^2 = \Omega(n)$ with probability $1 - o(1)$.

Theorem 12. *For any $\alpha \in \mathbb{R}^+$ and any $\delta > 0$, with probability at least $1 - e^{-\delta^2 n\mu_3^2/\sigma_3} - 2e^{-\delta^2 n/\sigma_2} - 2e^{-(n/4)p_{\mathcal{D},1}(\alpha)^2}$ over the choices of w_j's*

$$\mathsf{Inf}(f) \geqslant p_{\mathcal{D},1}(\alpha) \cdot \Theta(\alpha) \cdot \sqrt{n},$$

where $\Theta(\alpha) = \left(\alpha - \mu_3(1 + \frac{2\delta}{1-\delta})\right) / \sqrt{2\pi(1+\delta)}$, $\mu_3 = \mathbb{E}_{w\sim\mathcal{D}}[|w|^3]$, and σ_2 and σ_3 are the standard deviations of w^2 and $|w|^3$, respectively.

3.1 Proof of Theorem 12

As argued in Appendix A, it suffices to establish the lower bound on influence when $w_0 = 0$. To establish the theorem we argue similarly as in the proof of Lemma 8. We consider the case when the distribution $\mathcal{D}$ over $\mathbb{R}$ is symmetric around 0 and with variance 1. That is, $\mathbb{E}_{w\sim\mathcal{D}}[w^2] = 1$. Let $\mu_3 := \mathbb{E}_{w\sim\mathcal{D}}[|w|^3]$. Further, σ_2 and σ_3 denote the standard deviation of w^2 and $|w|^3$, respectively. That is, $\sigma_2^2 = \mathbb{E}[(w^2 - \mathbb{E}[w^2])^2]$ and $\sigma_3^2 = \mathbb{E}[(|w|^3 - \mathbb{E}[|w|^3])^2]$.

The following lemma gives a lower bound on the i-th influence.

Lemma 13. *Let $w_j \sim \mathcal{D}$ for $1 \leqslant j \leqslant n$. For all $i \in [n]$, $\alpha \in \mathbb{R}^+$, and $\delta > 0$, with probability at least $1 - e^{-\delta^2 n\mu_3^2/\sigma_3} - 2e^{-\delta^2 n/\sigma_2}$ over the choices of w_j's,*

$$\Pr_{x\in\{+1,-1\}^n}\left[\left|\sum_{1\leqslant j\leqslant n:\ j\neq i} w_j x_j\right| \leqslant \alpha\right] \geqslant \frac{\theta}{\sqrt{n}} - O\left(\frac{1}{n^{2/3}}\right),$$

where $\theta = (\alpha - \mu_3(1 + \frac{2\delta}{1-\delta}))/\sqrt{2\pi(1+\delta)}$.

Proof. To start we have $w_1, \ldots, w_n$ that are independently and identically distributed according to $\mathcal{D}$. Define random variables $A_i = (\sum_{j \neq i} |w_j|^3)/(n-1)$ and $B_i = (\sum_{j \neq i} w_j^2)/(n-1)$. Thus $\mathbb{E}[A_i] = \mu_3$ and $\mathbb{E}[B_i] = 1$. Using Bernstein's inequality (see Corollary 2.11 in Chapter 2 of [26]) on independent symmetric random variables w_j's we obtain

$$(1-\delta)(n-1) \leqslant \sum_{1 \leqslant j \leqslant n:\ j \neq i} w_j^2 \leqslant (1+\delta)(n-1), \tag{6}$$

with probability at least $1 - 2e^{-\delta^2 n/\sigma_2}$, where $\sigma_2^2 = \mathrm{Var}_{w \sim \mathcal{D}}(w^2)$. Similarly, we also have

$$\sum_{1 \leqslant j \leqslant n:\ j \neq i} w_j^3 \leqslant (1+\delta)C(n-1), \tag{7}$$

with probability at least $1 - e^{-2\delta^2 n\mu_3^2/\sigma_3}$, where $\sigma_3^2 = \mathrm{Var}_{w \sim \mathcal{D}}(|w|^3)$. Thus, from the union bound, it follows that both Eqs. (6) and (7), holds with probability at least $1 - e^{-\delta^2 n\mu_3^2/\sigma_3} - 2e^{-\delta^2 n/\sigma_2}$, and thus with this probability we have $(A_i/B_i) \leqslant \left(1 + \frac{2\delta}{1-\delta}\right)\mu_3$. Recall from Lemma 8 we know

$$\Pr_{x \in \{+1,-1\}^n}\left[\left|\sum_{j \leqslant n: j \neq i} w_j x_j\right| \leqslant \alpha\right] \geqslant \frac{1}{\sqrt{2\pi(n-1)B_i}}\left(\alpha - \frac{A_i}{B_i}\right) - O\left(\frac{1}{n^{2/3}}\right).$$

Thus plugging in the bound on (A_i/B_i) in the above inequality we obtain the lemma,

$$\Pr_{x \in \{+1,-1\}^n}\left[\left|\sum_{j \leqslant n: j \neq i} w_j x_j\right| \leqslant \alpha\right] \geqslant \frac{\alpha - (1 + \frac{2\delta}{1-\delta})\mu_3}{\sqrt{2\pi(1+\delta)(n-1)}} - O\left(\frac{1}{n^{2/3}}\right).$$

□

We now proceed to complete the proof of the theorem. For any $\alpha \in \mathbb{R}^+$,

$$\mathsf{Inf}(f) = \sum_i \mathsf{Inf}_i(f) \geqslant \sum_{i: w_i \geqslant \alpha} \mathsf{Inf}_i(f) \geqslant \sum_{i: w_i \geqslant \alpha} \Pr_{x \in \{+1,-1\}^n}\left[\left|\sum_{j \leqslant n:\ j \neq i} w_j x_j\right| \leqslant \alpha\right].$$

Thus, from Lemma 13, we have with probabiltity at least $1 - e^{-\delta^2 n\mu_3^2/\sigma_3} - 2e^{-\delta^2 n/\sigma_2}$ over the choices of w_j's,

$$\mathsf{Inf}(f) \geqslant N_\alpha \cdot \left(\frac{\alpha - \mu_3\left(1 + \frac{2\delta}{1-\delta}\right)}{\sqrt{2(n-1)\pi(1+\delta)}} - O\left(\frac{1}{n^{2/3}}\right)\right),$$

where N_α is the number of w_i's that are bigger than α. Now the following lemma shows that with probability at least $1 - 2e^{-\frac{n}{4}p_{\mathcal{D},1}(\alpha)^2}$ we have $N_\alpha \geqslant \frac{n}{2} \cdot p_{\mathcal{D},1}(\alpha)$, and that establishes the theorem.

Lemma 14. *Fix an $\alpha \in \mathbb{R}^+$. If we sample n points according to $\mathcal{D}$ then with probability at least $1 - 2e^{-\frac{n}{4}p_{\mathcal{D},1}(\alpha)^2}$ the number of points in the interval $[\alpha, \infty)$ is greater than $\frac{n \cdot p_{\mathcal{D},1}(\alpha)}{2}$ and less than $\frac{3n \cdot p_{\mathcal{D},1}(\alpha)}{2}$.*

Proof. Let $\mathbf{1}_i$ be the indicator function of the event that the i-th sample lies in the interval $[\alpha, \infty)$. Define $X = \sum_{j=1}^{n} \mathbf{1}_j$. Note that for each j, $\mathbb{E}[I_j] = p_{\mathcal{D},1}(\alpha)$, so using Chernoff bound on X we get,

$$\Pr\left[\frac{n \cdot p_{\mathcal{D},1}(\alpha)}{2} \leqslant X \leqslant \frac{3n \cdot p_{\mathcal{D},1}(\alpha)}{2}\right] \geqslant 1 - 2e^{-\frac{1}{4}n(p_{\mathcal{D},1}(\alpha))^2}.$$

□

3.2 Fourier Entropy Conjecture for a Random Linear Threshold Function

In this section, we give two natural examples of distributions on the w_i's under which the FEI conjecture holds with high probability. First we consider the uniform distribution on the closed interval $[-1, 1]$, and then the normal distribution $N(0, 1)$. Together this completes the proof of Theorem 1.

Uniform Distribution on $[-1, 1]$: $\mathcal{U}(-1, 1)$

Corollary 15. *If $w_i \sim \mathcal{U}(-1, 1)$ for $i \in \{0, \ldots, n\}$, then the FEI holds with high probability.*

Proof. Since $\mathbb{E}_{x \sim \mathcal{U}(-1,1)}[x^2] = 1/3$, when $w_0, \ldots, w_n$ are drawn independently from $\mathcal{U}[-1, 1]$, by Chernoff bound we have with high probability $\sum_{i=0}^{n} \mathbb{E}[w_i^2] = \Omega(n)$. Therefore, from Corollary 11, we have with high probability $\mathsf{Inf}(f) = \Omega(\sqrt{n})$. Using the upper bound on the Fourier entropy from Theorem 2 we have our result. □

Normal Distribution: $N(0, 1)$. We note that for the normal distribution, unlike the case of uniform distribution, the FEI conjecture does not follow from Khintchine's inequality. Indeed, Corollary 11 does not give us what we want, i.e., a high probability on the event $\mathsf{Inf}(f) = \Omega(\sqrt{n})$. On the other hand, if we try to boost the probability we end up with a weaker lower bound on the influence. To see this, observe that when $w_1, \ldots, w_n$ are drawn independently from $N(0, 1)$,

1. $\mathbb{E}[\sum_i w_i^2] = n$ (since $\mathbb{E}[X^2] = 1$ for $X \sim N(0, 1)$). This implies that with high probability $\sum w_i^2 = \Omega(n)$, and hence $\sqrt{\sum w_i^2} = \Omega(\sqrt{n})$.
2. The probability that $\max_{i=1}^{n} |w_i| = O(\sqrt{\log n})$ is $1 - o(1)$. Using the fact $\Pr[X > x] \leqslant e^{-x^2/2}$ when $X \sim N(0, 1)$, we have $\Pr[w_i > \sqrt{2c \log n}] \leqslant \frac{1}{n^c}$. Since w_i's are drawn independently, the probability that all the w_i's are less than $\sqrt{2c \log n}$ is lower bounded by $1 - \frac{1}{n^{c-1}}$, where $c > 1$ is a constant.

Applying Corollary 11 to this now gives us $\mathsf{Inf}(f) \geqslant \Omega\left(\sqrt{\frac{n}{\log n}}\right)$ with $1 - o(1)$ probability. Moreover, the max-central limit theorem implies that for large enough n, with high probability, $\max_i |w_i| = \Theta(\sqrt{\log n})$ (see Example 10.5.3 in [27]). However using the other technique, namely Theorem 12, we can obtain our desired result.

Corollary 16. *If $w_i \sim N(0,1)$ for $i \in \{0,\ldots,n\}$, then the FEI holds with probability at least $1 - e^{-\Omega(n)}$.*

Proof. We use the following well known bounds on moments of $N(0,1)$.

$$\mathbb{E}_{w\sim N(0,1)}[|w|^r] = \begin{cases} \sqrt{\frac{2}{\pi}}\,(r-1)!! & \text{if } r \text{ is odd,} \\ (r-1)!! & \text{if } r \text{ is even.} \end{cases}$$

Therefore, we have $\mu_3 = \frac{2\sqrt{2}}{\sqrt{\pi}}$, $\sigma_2^2 = \mathbb{E}[w^4] - (\mathbb{E}[w^2])^2 = 2$, and $\sigma_3^2 = \mathbb{E}[|w|^6] - (\mathbb{E}[|w|^3])^2 = (15 - \frac{8}{\pi})$. Further, we set $\alpha = \mu_3(2 + \frac{2\delta}{1-\delta})$, for any $\delta > 0$. Hence,

$$p_{\mathcal{D},1}(\alpha) \geqslant 2 \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{\alpha^2}{2}} \left(\frac{1}{\alpha} - \frac{1}{\alpha^3}\right) = \Omega(1).$$

The first inequality is easily established; for example, see Exercise 1, Chapter 7 in [28]. Thus, from Theorem 12 we have that $\mathsf{Inf}(f) = \Omega(\sqrt{n})$ and now using Theorem 2 we have our result. □

Remark 1. We note that the proof of Theorem 12 shows that the FEI conejcture holds with high probability, as long as $\mathcal{D}$ (with $\mu = 0$ and $\sigma^2 = 1$) satisfies the following properties: (i) $\mathbb{E}_{w\sim\mathcal{D}}[|w|^3]$ is finite, and (ii) $\Pr_{w\sim\mathcal{D}}[|w| > \mathbb{E}_{w\sim\mathcal{D}}[|w|^3]] = \Omega(1)$. In particular, our proof holds for *truncated* distributions, and the optimality of the constant in Shevtsova's theorem (Theorem 7) is crucial in such cases to establish property (ii).

4 Conclusion and Open Problems

We proved in this paper that for a random linear threshold function $f(x) := \mathsf{sign}(w_0 + \sum_i w_i x_i)$, where the w_i's are drawn from a distribution that has some 'nice' properties (Remark 1), the FEI conjecture holds with high probability. Moreover, we show that the uniform distribution over an interval, say $[-1,1]$, and the normal distribution $N(0,1)$ satisfy these 'nice' properties. Indeed, we established that a random LTF sampled according to these distributions has influence $\Omega(\sqrt{n})$. When combined with the $O(\sqrt{n})$ upper bound (Theorem 2) from previous work on the Fourier entropy of *all* LTF's, we conclude that the FEI conjecture holds for the random LTF's sampled as above. In the process, we obtain non-trivial lower bounds on the influence of f in terms of the w_i's.

An obvious open question is to prove that the FEI conjecture holds for all LTF's. While our current techniques seem far from sufficient to achieve this goal, a natural question is to prove a generalization of Theorem 2, and give an upper bound on the Fourier entropy in terms of w_i's. We believe it is possible to obtain such a general upper bound as a function of a suitable notion of "skewness" of the weights w_i.

Another natural question is whether one can show that FEI conjecture holds for a random *polynomial* threshold function of degree d. A polynomial threshold function of degree d is defined as the sign of a degree-d polynomial. For a degree-d PTF $f(x) = \mathsf{sign}(p(x_1, \ldots, x_n))$, observe that $\mathsf{Inf}_i(f) = \Pr_x[\mathsf{sign}(p(x)) \neq \mathsf{sign}(p(x^i))] = \Pr_x[|p(x)| < 2|D_i p(x)|]$, where $D_i p(x)$ is the partial derivative of $p(x)$ with respect to x_i. Hence, to lower bound the influence, a standard approach would be to show that $p(x)$ has certain *concentration* properties and $D_i p(x)$ has certain *anti-concentration* properties, for a random p. However, we currently do not know such strong enough concentration/anti-concentration bounds. Our techniques do not seem to generalize even to polynomial threshold functions of degree 2. Such generalizations seem to require more powerful tools. We also note that currently the best bound on the Fourier entropy of a degree-d PTF is $O(\sqrt{n}(\log n)^{O(d \log d)})$ where the constant in $O(\cdot)$ depends on the degree d. (It follows from Kane's bound on the average sensitivity of a PTF [29].)

Acknowledgments. We thank the reviewers for helpful comments that improved the presentation of the paper.

A Reducing Non-homogeneous to Homogeneous case

Let $f(x) = \mathsf{sign}(w_0 + \sum_{i=1}^n w_i x_i)$ for all $x \in \{+1, -1\}^n$. Recall from Eq. (2) we have an exact expression for the i-th influence for all $1 \leqslant i \leqslant n$. We can relax the probability estimate to lower bound the influence as follows,

$$\mathsf{Inf}_i(f) \geqslant \frac{1}{2} \Pr_{x \in \{+1,-1\}^n} \left[\left| w_0 + \sum_{j \leqslant n:\, j \neq i} w_j x_j \right| \leqslant |w_i| \right]. \tag{8}$$

Now consider the function $g(x_0, x_1, \ldots, x_n) = \mathsf{sign}(\sum_{i=0}^n w_i x_i)$ by adding the extra variable x_0. We claim that $\mathsf{Inf}_i(g) \leqslant 2\mathsf{Inf}_i(f)$, for all $1 \leqslant i \leqslant n$. Fix an $i \in [n]$. From Eq. (2) we know that $\mathsf{Inf}_i(g)$ equals

$$\frac{1}{2}\left(\Pr\left[-|w_i| < w_0 + \sum_{1 \leqslant j \leqslant n:\, j \neq i} w_j x_j \leqslant |w_i| \right] + \Pr\left[-|w_i| < -w_0 + \sum_{1 \leqslant j \leqslant n:\, j \neq i} w_j x_j \leqslant |w_i| \right] \right),$$

where the probabilities are uniform distribution over $x_1, \ldots, x_n \in \{+1, -1\}^n$. By relaxing the event in each case we have

$$\mathsf{Inf}_i(g) \leqslant \frac{1}{2}\left(\Pr\left[-|w_i| \leqslant w_0 + \sum_{1 \leqslant j \leqslant n:\, j \neq i} w_j x_j \leqslant |w_i| \right] + \Pr\left[-|w_i| \leqslant -w_0 + \sum_{1 \leqslant j \leqslant n:\, j \neq i} w_j x_j \leqslant |w_i| \right] \right).$$

It is easily seen that,

$$\Pr_{x\in\{+1,-1\}^n}\left[-|w_i| \leqslant w_0 + \sum_{1\leqslant j\leqslant n:\, j\neq i} w_j x_j \leqslant |w_i|\right]$$
$$= \Pr_{x\in\{+1,-1\}^n}\left[-|w_i| \leqslant -w_0 + \sum_{1\leqslant j\leqslant n:\, j\neq i} w_j x_j \leqslant |w_i|\right] .$$

Indeed, there exists a 1-1 correspondence between n-bit strings satisfying the left hand side event and the right hand side event. Thus,

$$\mathsf{Inf}_i(g) \leqslant \Pr_{x\in\{+1,-1\}^n}\left[-|w_i| \leqslant w_0 + \sum_{1\leqslant j\leqslant n:\, j\neq i} w_j x_j \leqslant |w_i|\right] \leqslant 2\cdot \mathsf{Inf}_i(f) \ ,$$

where the second inequality follows from (8).

Therefore, we have

$$\mathsf{Inf}(f) \geqslant \sum_{i=1}^{n} \frac{\mathsf{Inf}_i(g)}{2} = \frac{\mathsf{Inf}(g) - \mathsf{Inf}_0(g)}{2} \geqslant \frac{\mathsf{Inf}(g) - 1}{2}.$$

Thus we can translate a lower bound on influences in the homogeneous case to the non-homogeneous case.

References

1. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform, and learnability. J. ACM **40**(3), 607–620 (1993)
2. Boppana, R.B.: The average sensitivity of bounded-depth circuits. Inf. Process. Lett. **63**(5), 257–261 (1997)
3. Ganor, A., Komargodski, I., Lee, T., Raz, R.: On the noise stability of small Demorgan formulas, Technical report, Electronic Colloquium on Computational Complexity (ECCC) TR 12-174 (2012)
4. O'Donnell, R., Saks, M., Schramm, O.: Every decision tree has an influential variable. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005, pp. 31–39. IEEE Computer Society (2005)
5. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014)
6. Friedgut, E., Kalai, G.: Every monotone graph property has a sharp threshold. Proc. Am. Math. Soc. **124**(10), 2993–3002 (1996)
7. Bourgain, J., Kalai, G.: Influences of variables and threshold intervals under group symmetries. Geom. Funct. Anal. (GAFA) **7**(3), 438–461 (1997)
8. Mansour, Y.: An $\mathcal{O}(\mathrm{n}^{\log\log n})$ learning algorithm for DNF under the uniform distribution. J. Comput. Syst. Sci. **50**(3), 543–550 (1995)
9. Gopalan, P., Kalai, A.T., Klivans, A.: Agnostically learning decision trees. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 527–536 (2008)

10. Gopalan, P., Kalai, A., Klivans, A.R.: A query algorithm for agnostically learning DNF? In: 21st Annual Conference on Learning Theory - COLT 2008, 9–12 July 2008, Helsinki, Finland, pp. 515–516 (2008)
11. Friedgut, E.: Boolean functions with low average sensitivity depend on few coordinates. Combinatorica **18**(1), 27–35 (1998)
12. Kahn, J., Kalai, G., Linial, N.: The influence of variables on Boolean functions. In: Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 68–80 (1988)
13. O'Donnell, R., Wright, J., Zhou, Y.: The Fourier entropy–influence conjecture for certain classes of Boolean functions. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 330–341. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_28
14. Kalai, G.: The entropy/influence conjecture. Terence Tao's blog: https://terrytao.wordpress.com/2007/08/16/gil-kalai-the-entropyinfluence-conjecture/
15. Klivans, A., Lee, H., Wan, A.: Mansour's conjecture is true for random DNF formulas. In: Proceedings of the 23rd Conference on Learning Theory, pp. 368–380 (2010)
16. Das, B., Pal, M., Visavaliya, V.: The entropy influence conjecture revisited. Technical report, arXiv:1110.4301 (2011)
17. O'Donnell, R., Tan, L.Y.: A composition theorem for the Fourier entropy-influence conjecture. In: Proceedings of Automata, Languages and Programming - 40th International Colloquium, pp. 780–791 (2013)
18. Chakraborty, S., Kulkarni, R., Lokam, S.V., Saurabh, N.: Upper bounds on Fourier entropy. Theor. Comput. Sci. **654**, 92–112 (2016)
19. Wan, A., Wright, J., Wu, C.: Decision trees, protocols and the entropy-influence conjecture. In: Innovations in Theoretical Computer Science, pp. 67–80 (2014)
20. Hicks, J.S., Wheeling, R.F.: An efficient method for generating uniformly distributed points on the surface of an n-dimensional sphere. Commun. ACM **2**(4), 17–19 (1959)
21. Muller, M.E.: A note on a method for generating points uniformly on n-dimensional spheres. Commun. ACM **2**(4), 19–20 (1959)
22. Marsaglia, G.: Choosing a point from the surface of a sphere. Ann. Math. Stat. **43**(2), 645–646 (1972)
23. Petersen, W.P., Bernasconi, A.: Uniform sampling from an n-sphere. Technical report. Swiss Center for Scientific Computing (1997)
24. Szarek, S.J.: On the best constants in the Khinchine inequality. Studia Math. **58**, 197–208 (1976)
25. Shevtsova, I.: Moment-type estimates with asymptotically optimal structure for the accuracy of the normal approximation. Annales Mathematicae Et Informaticae **39**, 241–307 (2012)
26. Boucheron, S., Lugosi, G., Massart, P.: Concentration Inequalities: A Nonasymptotic Theory of Independence. Oxford University Press, Oxford (2013)
27. David, H.A., Nagaraja, H.N.: Order Statistics, 3rd edn. Wiley, Hoboken (2003)
28. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1, 3rd edn. Wiley, Hoboken (1968)
29. Kane, D.M.: The correct exponent for the Gotsman-Linial Conjecture. Comput. Complex. **23**(2), 151–175 (2014)

Property Suffix Array with Applications

Panagiotis Charalampopoulos, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis(✉)

Department of Informatics, King's College London, London, UK
{panagiotis.charalampopoulos,costas.iliopoulos,chang.2.liu, solon.pissis}@kcl.ac.uk

Abstract. The suffix array is one of the most prevalent data structures for string indexing; it stores the lexicographically sorted list of suffixes of a given string. Its practical advantage compared to the suffix tree is space efficiency. In *Property Indexing*, we are given a string x of length n and a property Π, i.e. a set of *Π-valid* intervals over x. A suffix-tree-like index over these valid prefixes of suffixes of x can be built in time and space $\mathcal{O}(n)$. We show here how to directly build a suffix-array-like index, the *Property Suffix Array* (PSA), in time and space $\mathcal{O}(n)$. We mainly draw our motivation from weighted (probabilistic) sequences: sequences of probability distributions over a given alphabet. Given a probability threshold $\frac{1}{z}$, we say that a string p of length m matches a weighted sequence X of length n at starting position i if the product of probabilities of the letters of p at positions $i, \ldots, i+m-1$ in X is at least $\frac{1}{z}$. Our algorithm for building the PSA can be directly applied to build an $\mathcal{O}(nz)$-sized suffix-array-like index over X in time and space $\mathcal{O}(nz)$.

1 Introduction

Property matching, introduced in [4], comprises of matching a pattern to a text of which only certain intervals are valid. The on-line version of this problem is trivial and thus the indexing version has received much more attention. In the *Property Indexing* problem, we are given a text x of length n over an alphabet of size σ and a *property Π*; Π is a set of subintervals of $[0, n-1]$ with integer endpoints. The goal is to then preprocess the text so that given a pattern p we can return its occurrences in the Π-valid intervals of x, i.e. we want to report $x[i \mathinner{.\,.} j]$ if and only if it is equal to p and $[i, j]$ is a subinterval of some $[a, b] \in \Pi$.

Most of the prevalent text indexing data structures are built over the suffixes of the text [22]. However, by introducing the property Π only some prefixes of each suffix are now valid. The authors in [4] presented an algorithm for building the *Property Suffix Tree* (PST) in $\mathcal{O}(n \log \sigma + n \log\log n)$ time for integer alphabets, implicitly sorting the prefixes of the suffixes that are valid. Recently, the authors in [5,6] have presented an $\mathcal{O}(n)$-time algorithm for the construction

P. Charalampopoulos—Supported by the Graduate Teaching Scholarship scheme of the Department of Informatics at King's College London and by the A.G. Leventis Foundation.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 290–302, 2018.
https://doi.org/10.1007/978-3-319-77404-6_22

of the PST that also works for integer alphabets. This is based on a technique by Kociumaka, Radoszewski, Rytter and Waleń for answering off-line weighted ancestor queries in suffix trees (see the Appendix of [5]). A dynamic instance of Property Indexing has also been studied in [19], where the author also makes use of the suffix tree.

An $\mathcal{O}(n)$-time algorithm for building an index over the suffix tree of x for integer alphabets that allows for property matching queries was proposed by the authors of [14,15]. This solution, however, does not sort the prefixes of suffixes that are valid (which is an interesting problem per se); it offloads the difficulty of the computation from the construction to the queries.

The *suffix array* (SA) of a text x of length n is an integer array of size n that stores the lexicographically sorted list of suffixes of x [20]. In order to construct the *Property Suffix Array*, which we denote by PSA, we essentially need to lexicographically sort a multiset consisting of substrings of x; this multiset contains at most one prefix of each suffix of x. This can be achieved in linear time by traversing the PST, however our aim here is to do it directly—we do not want to construct or store the PST. It is well-known from the setting of standard strings that the SA is more space efficient than the suffix tree [1].

Note that for clarity of presentation we represent Π—and assume the input is given in this form—by an integer array $\mathcal{L}$ of size n, such that

$$\mathcal{L}[i] = \max\{j|(k,j) \in \Pi, k \leq i\} - i + 1$$

is the length of the longest prefix of $x[i\mathinner{.\,.}n-1]$ that is valid. It should be clear that $\mathcal{L}$ can be obtained from Π in $\mathcal{O}(n+|\Pi|)$ time. We also assume that $\mathcal{L}[i] > 0$ for all i; the case that $\mathcal{L}[i] = 0$ can be handled easily as the resulting substring would just be the empty string.

Example 1 (Running example). Consider the string $x =$ `acababaab` and property $\Pi = \{(0,3),(4,6),(6,8)\}$:

i	0	1	2	3	4	5	6	7	8
$x[i]$	a	c	a	b	a	b	a	a	b
$\mathcal{L}[i]$	4	3	2	1	3	2	3	2	1
SA$[i]$	6	7	4	2	0	8	5	3	1
PSA$[i]$	6	2	7	4	0	3	8	5	1

Our main result is an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space direct construction of the PSA for integer alphabets. The problem can be formally defined as follows.

PROPERTY SUFFIX ARRAY
Input: A string x of length n and an integer array $\mathcal{L}$ of size n, satisfying $0 < \mathcal{L}[i] \leq n-i$ and $\mathcal{L}[i] \geq \mathcal{L}[i-1]-1$.
Output: An array PSA that stores a permutation of $0,\ldots,n-1$ and for all $1 \leq r < n$, letting PSA$[r-1] = j$ and PSA$[r] = k$, we have $x[j\mathinner{.\,.}j+\mathcal{L}[j]-1] \leq x[k\mathinner{.\,.}k+\mathcal{L}[k]-1]$.

Application. We apply our solution to this problem in the setting of weighted sequences. In a weighted sequence every position contains a subset of the alphabet and every letter of this subset is associated with a probability of occurrence such that the sum of probabilities at each position equals 1. This data representation is common in a wide range of applications: (i) imprecise sensor data measurements; (ii) flexible sequence modeling, such as binding profiles of DNA sequences; (iii) observations that are private and thus sequences of observations may have artificial uncertainty introduced deliberately (see [2] for a survey). Pattern matching (or substring matching) is a core operation in a wide variety of applications including bioinformatics, information retrieval, text mining, and pattern recognition. Many pattern matching applications generalize naturally to the weighted case as much of this data is more commonly uncertain (e.g. genomes with incorporated SNPs from a population) than certain.

In the *weighted pattern matching* (WPM) problem we are given a string p of length m called a pattern, a weighted sequence X of length n called a text, both over an alphabet Σ of size σ, and a *threshold probability* $\frac{1}{z}$. The task is to find all positions i in X where the product of probabilities of the letters of p at positions $i, \dots, i+m-1$ in X is at least $\frac{1}{z}$ [8,17]. Each such position is called an *occurrence* of the pattern; we also say that the fragment and the pattern *match*.

Here we consider the problem of indexing a weighted sequence. We are given a weighted sequence X of length n and a probability threshold $\frac{1}{z}$, and we are asked to construct an index which will allow us to efficiently answer queries with respect to the contents of X. This problem was considered in [4], where a reduction to Property Indexing of a text of size $\mathcal{O}(nz^2 \log z)$ was proposed. The authors in [6] reduced this to a text of size nz, thus presenting an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space construction of an $\mathcal{O}(nz)$-sized index that answers pattern matching queries on X in optimal time. The same index as the one in [6] was first presented in [7] but with a different $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space construction algorithm. Approximate variants of these indexes have also been considered in [6,10].

All these indexes [4,6,7] are based on constructing and traversing the suffix tree. Here, using our solution to problem Property Suffix Array *and* the main idea of [6], we show how to construct directly an array data structure within the same complexities. Moreover, we present experiments that show the advantage of our new data structure: as expected, it requires much less space than the one of [6,7]. Our index, apart from being *simple* and *small* in practice, is *asymptotically smaller* than the input weighted sequence when $z = o(\sigma)$.

Structure of the paper. In Sect. 3, we provide three $\mathcal{O}(n)$-space algorithms for computing the PSA directly, with time complexities $\mathcal{O}(n \log^2 n)$, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n)$. In Sect. 4, we apply our solution to this general problem in the setting of weighted sequences to obtain an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space algorithm for constructing a new $\mathcal{O}(nz)$-sized array index for weighted sequences. Finally, in Sect. 5, we present an experimental evaluation of the proposed algorithms.

2 Preliminaries

Let $x = x[0]x[1]\dots x[n-1]$ be a *string* of length $|x| = n$ over a finite ordered *alphabet* Σ of size σ, i.e. $\sigma = |\Sigma|$. In particular, we consider the case of an *integer alphabet*; in this case each letter is replaced by its rank such that the resulting string consists of integers in the range $\{1, \dots, n\}$.

For two positions i and j on x, we denote by $x[i\,..\,j] = x[i]\dots x[j]$ the *factor* (sometimes called *substring*) of x that starts at position i and ends at position j. We recall that a *prefix* of x is a factor that starts at position 0 ($x[0\,..\,j]$) and a *suffix* of x is a factor that ends at position $n-1$ ($x[i\,..\,n-1]$). We denote a string x that is lexicographically smaller than (resp. smaller than or equal to) a string y by $x < y$ ($x \leq y$).

2.1 Suffix Array

We denote by SA the *suffix array* of a non-empty string x of length n. SA is an integer array of size n storing the starting positions of all (lexicographically) sorted non-empty suffixes of x, i.e. for all $1 \leq r < n$ we have $x[\mathsf{SA}[r-1]\,..\,n-1] < x[\mathsf{SA}[r]\,..\,n-1]$ [20]. Let $\mathsf{lcp}(r, s)$ denote the length of the longest common prefix between $x[\mathsf{SA}[r]\,..\,n-1]$ and $x[\mathsf{SA}[s]\,..\,n-1]$ for all positions r, s on x, and 0 otherwise. We denote by LCP the *longest common prefix* array of y defined by $\mathsf{LCP}[r] = \mathsf{lcp}(r-1, r)$ for all $1 \leq r < n$, and $\mathsf{LCP}[0] = 0$. The inverse iSA of the array SA is defined by $\mathsf{iSA}[\mathsf{SA}[r]] = r$, for all $0 \leq r < n$. It is known that SA [21], iSA, and LCP [16] of a string of length n, over an integer alphabet, can be computed in time and space $\mathcal{O}(n)$. It is then known that a range minimum query (RMQ) data structure over the LCP array, that can be constructed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space [9], can answer lcp queries in $\mathcal{O}(1)$ time per query by returning the index of a minimal value in the respective range of the SA.

3 $\mathcal{O}(n)$-Space Algorithms for Computing PSA

3.1 Sparse Table-Based $\mathcal{O}(n \log^2 n)$-Time Algorithm

The algorithm presented in this subsection applies a combination of the *Sparse Table* idea for answering RMQs [9] and the *doubling technique* [20] to the context of sorting prefixes of suffixes (factors) of x. Using this combination, one may easily obtain an $\mathcal{O}(n \log n)$-time and $\mathcal{O}(n \log n)$-space algorithm for constructing the PSA [12]. We tweak this solution to require only $\mathcal{O}(n)$ space, suffering an additional multiplicative $\log n$ factor in the time complexity. There are $\mathcal{O}(\log n)$ levels: at the kth level, we sort prefixes of suffixes up to length 2^{k+1}; at each level, $\mathcal{O}(n \log n)$ time is required to sort these factors using any optimal comparison-based sorting algorithm [11].

The aforementioned scheme assumes that we can compare two factors in constant time. To this end, we borrow the Sparse Table algorithm idea for answering RMQs: the minimum value in a given range r is the minimum between the minimums of any two, potentially overlapping, subranges whose union is r. The same idea can be applied in a completely different context:

Fact 2. *Given two strings x and y, with $|x| \leq |y|$, and $k = \lfloor \log |x| \rfloor$, $x \leq y$ if and only if* $(x[0\,..\,2^k], x[|x|-2^k\,..\,|x|-1]) \leq (y[0\,..\,2^k], y[|x|-2^k\,..\,|x|-1])$.

We thus compute the ranks of prefixes of suffixes whose lengths are multiples of two using the doubling technique [20] and then use these ranks to sort prefixes whose lengths may not be multiples of two by applying Fact 2. Note that this computation can be done level by level in a total of $\mathcal{O}(\log n)$ levels, and therefore the working space is $\mathcal{O}(n)$. We formalize this algorithm, denoted by ST-PSA, in the pseudocode below. We start by initializing the elements in the PSA by sorting and ranking the letters of x (Lines 2–8). We store these ranks in an array (Line 9). Then, at level k (Line 10), we compute the ranks of prefixes whose lengths are multiples of two using the previous level information and radix sort in $\mathcal{O}(n)$ time (Lines 11–12). Next, we sort and rank *all* prefixes up to length 2^{k+1} using a comparison-based sorting algorithm and Fact 2 in $\mathcal{O}(n \log n)$ time (Lines 13–14). We store these ranks in an array (Line 15) and proceed to the next level. Thus the total time required is $\mathcal{O}(n \log^2 n)$ and the space is $\mathcal{O}(n)$. The value of this algorithm is its practicality: (a) it requires very little space; (b) the number of levels required is in fact $\lfloor \log \ell \rfloor$, where ℓ is the maximum value in $\mathcal{L}$; and (c) at level k it suffices to sort groups of elements having the same rank at level $k-1$.

```
1  Algorithm ST-PSA($x, n, \mathcal{L}$)
2      for $i \leftarrow 0$ to $n-1$ do
3          PSA[$i$] $\leftarrow i$;
4      Sort PSA using the following comparison rule for PSA[$i$] and PSA[$j$]:
5          if $x[i] < x[j]$ then PSA[$i$] < PSA[$j$];
6          else if $x[i] > x[j]$ then PSA[$i$] > PSA[$j$];
7          else PSA[$i$] = PSA[$j$];
8      Rank the elements of PSA and store their ranks in $\mathsf{Rank}_{\mathsf{PSA}}$;
9      $\mathsf{Rank}_{\mathsf{PREF}} \leftarrow \mathsf{Rank}_{\mathsf{PSA}}$;
10     for $k \leftarrow 1$ to $\lfloor \log n \rfloor$ do
11         Construct an array $\mathcal{A}$ of pairs: $\mathcal{A}[i] = (\mathsf{Rank}_{\mathsf{PREF}}[i], \mathsf{Rank}_{\mathsf{PREF}}[i + 2^{k-1}])$;
12         Sort the pairs in $\mathcal{A}$ using radix sort and store their ranks in $\mathsf{Rank}_{\mathsf{CURR}}$;
13         Sort PSA using $\mathcal{L}$, $\mathsf{Rank}_{\mathsf{PSA}}$, $\mathsf{Rank}_{\mathsf{CURR}}$, and Fact 2 for the comparison;
14         Rank the elements of PSA and store their new ranks in $\mathsf{Rank}_{\mathsf{PSA}}$;
15         $\mathsf{Rank}_{\mathsf{PREF}} \leftarrow \mathsf{Rank}_{\mathsf{CURR}}$;
16     return PSA;
```

3.2 LCP-Based $\mathcal{O}(n \log n)$-Time Algorithm

The algorithm presented in this subsection is based on the following fact.

Fact 3. *Given two factors of x, $x[i_1\,..\,j_1]$ and $x[i_2\,..\,j_2]$, with $\mathit{iSA}[i_1] < \mathit{iSA}[i_2]$, we have that $x[i_2\,..\,j_2] \leq x[i_1\,..\,j_1]$ if and only if $j_2 - i_2 \leq \mathit{lcp}(\mathit{iSA}[i_1], \mathit{iSA}[i_2])$ and $j_2 - i_2 \leq j_1 - i_1$.*

Recall that lcp queries for two arbitrary suffixes of x can be answered in time $\mathcal{O}(1)$ per query after an $\mathcal{O}(n)$-time preprocessing of the LCP array of x [9, 20]. We can then perform any optimal comparison-based sorting algorithm (use Fact 3 for the comparison) on the set of prefixes of suffixes. Thus the total time required is $\mathcal{O}(n \log n)$ and the working space is $\mathcal{O}(n)$. We formalize this algorithm, denoted by LCP-PSA, in the pseudocode below.

```
Algorithm LCP-PSA($x, n, \mathcal{L}$)
    Compute $\mathsf{SA}, \mathsf{iSA}, \mathsf{LCP}, \mathsf{RMQ}_{\mathsf{LCP}}$ of $x$;
    for $i \leftarrow 0$ to $n-1$ do
        $\mathsf{PSA}[i] \leftarrow \mathsf{SA}[i]$;
    Sort PSA using the following comparison rule for $\mathsf{PSA}[i]$ and $\mathsf{PSA}[j]$:
        if $i < j$ then $k \leftarrow \mathsf{RMQ}_{\mathsf{LCP}}(i+1, j)$;
        else $k \leftarrow \mathsf{RMQ}_{\mathsf{LCP}}(j+1, i)$;
        if $LCP[k] < \min\{\mathcal{L}[SA[i]], \mathcal{L}[SA[j]]\}$ then
            $\mathsf{PSA}[i] < \mathsf{PSA}[j]$;
        else
            if $\mathcal{L}[SA[i]] < \mathcal{L}[SA[j]]$ then
                $\mathsf{PSA}[i] < \mathsf{PSA}[j]$;
            else
                $\mathsf{PSA}[i] > \mathsf{PSA}[j]$;
    return PSA;
```

3.3 Union-Find-Based $\mathcal{O}(n)$-Time Algorithm

In this section we assume the precomputation of SA, iSA and LCP of x. Given the iSA, the LCP array and $\mathcal{L}$, let $f_i = \max_{0 \le r \le \mathsf{iSA}[i]} \{r | \mathsf{LCP}[r] < \mathcal{L}[i]\}$. Informally, f_i tells us how many suffixes are lexicographically smaller than $x[i \mathinner{.\,.} i + \mathcal{L}[i] - 1]$ (see also Example 5 in this regard). It follows from the following lemma that in order to construct the PSA it is enough to sort the ordered pairs $(f_i, \mathcal{L}[i])$.

Lemma 4. *Given two factors of x, $x[i_1 \mathinner{.\,.} j_1]$ and $x[i_2 \mathinner{.\,.} j_2]$, we have that if $(f_{i_1}, j_1 - i_1) \le (f_{i_2}, j_2 - i_2)$ then $x[i_1 \mathinner{.\,.} j_1] \le x[i_2 \mathinner{.\,.} j_2]$.*

Proof. Note that $x[i \mathinner{.\,.} j]$ is a prefix of $x[\mathsf{SA}[f_i] \mathinner{.\,.} n-1]$. Thus if

- either $f_{i_1} < f_{i_2}$
- or $f_{i_1} = f_{i_2}$ and $j_1 - i_1 \le j_2 - i_2$

then we have that $x[i_1 \mathinner{.\,.} j_1] \le x[i_2 \mathinner{.\,.} j_2]$. □

Example 5 (Running example).

i	0	1	2	3	4	5	6	7	8
$\mathcal{L}[i]$	4	3	2	1	3	2	3	2	1
$\mathsf{SA}[i]$	6	7	4	2	0	8	5	3	1
$\mathsf{LCP}[i]$	0	1	2	3	1	0	1	2	0
$\mathcal{L}[\mathsf{SA}[i]]$	3	2	3	2	4	1	2	1	3
$f_{\mathsf{SA}[i]}$	0	1	2	1	4	5	6	5	8
$\mathsf{PSA}[i]$	6	2	7	4	0	3	8	5	1

For $i = 3$, we have that $\mathsf{iSA}[3] = 7$, and hence we obtain the pair $(f_3, \mathcal{L}[3]) = (5, 1)$.

The computational problem is to compute f_i efficiently for all i; for this we rely on the Union-Find data structure [11] in a similar manner as the authors in [18]. Our technique also resembles the technique by Kociumaka, Radoszewski, Rytter and Waleń for answering off-line weighted ancestor queries in trees; it can be found in the Appendix of [5]. Union-Find maintains a partition of $\{0, 1, \ldots, n-1\}$, where each set has a representative element, and supports three basic operations:

– $\mathsf{MakeSet}(n)$ creates n new sets $\{0\}, \{1\}, \ldots, \{n-1\}$, where the representative index of set $\{i\}$ is i.
– $\mathsf{Find}(i)$ returns the representative of the set containing i.
– $\mathsf{Union}(i, j)$ first finds the set S_i containing i and the set S_j containing j. If $S_i \neq S_j$, then they are replaced by the set $S_i \cup S_j$.

In the algorithm described below, we only encounter *linear* Union-Find instances, in which the sets of the partition consist of consecutive integers and the representative of each set is its smallest element. We rely on the following result.

Theorem 6 ([13]). *A sequence of q given linear Union and Find operations over a partition of $\{0, 1, \ldots, n-1\}$ can be performed in time $\mathcal{O}(n + q)$.*

We perform the following initialization steps in $\mathcal{O}(n)$ time:

1. Initialize an array $\mathcal{A}$ of linked lists of size n;
2. Initialize the Union-Find data structure by calling $\mathsf{MakeSet}(n)$;
3. Sort indices $\{0, 1, \ldots, n-1\}$ based on $\mathcal{L}[i]$ (store them in an array $\mathcal{M}_{\mathcal{L}}$);
4. Sort indices $\{0, 1, \ldots, n-1\}$ based on $\mathsf{LCP}[i]$ (store them in an array $\mathcal{M}_{\mathsf{LCP}}$).

Then, for all j from $k = \max\{\max_i\{\mathsf{LCP}[i]\}, \max_i\{\mathcal{L}[i]\}\}$ down to 1 we do the following:

1. $\mathsf{Union}(i-1, i)$ for each i such that $\mathsf{LCP}[i] = j$ using $\mathcal{M}_{\mathsf{LCP}}$;
2. We find all i for which $\mathcal{L}[i] = j$ using $\mathcal{M}_{\mathcal{L}}$ and conclude that $f_i = \mathsf{Find}(\mathsf{iSA}[i])$; we store i at the head of the linked list $\mathcal{A}[f_i]$.

Note that after performing the Union operations for some j, the representative element of the set containing α, $\mathsf{Find}(\alpha)$, is the greatest $\beta \leq \alpha$, for which $\mathsf{LCP}[\beta] \leq j - 1$. Thus, in the end of the computation, $\mathcal{A}[j]$ stores the indices i, for which $f_i = j$. In addition, the elements of each list $\mathcal{A}[j]$ are in the order of non-decreasing $\mathcal{L}[i]$. We can thus just read the elements of the linked lists in $\mathcal{A}$ from the left to the right and from the head to the tail to obtain the PSA. We formalize this algorithm, denoted by UF-PSA, in the pseudocode below.

```
Algorithm UF-PSA($x, n, \mathcal{L}$)
    Compute SA, iSA and LCP of $x$;
    Construct a map $\mathcal{M}_{\mathsf{LCP}}$ such that $\mathcal{M}_{\mathsf{LCP}}[i] = \{j \mid \mathsf{LCP}[j] = i\}$;
    Construct a map $\mathcal{M}_{\mathcal{L}}$ such that $\mathcal{M}_{\mathcal{L}}[i] = \{j \mid \mathcal{L}[j] = i\}$;
    Initialize an array of lists $\mathcal{A}$ of size $n$;
    Initialize a Union-Find data structure $\mathcal{UF}$;
    $\mathcal{UF}$.MakeSet($n$);
    $lcp_{\max} \leftarrow \max\{\mathsf{LCP}[0], \mathsf{LCP}[1], \ldots, \mathsf{LCP}[n-1]\}$;
    $\ell_{\max} \leftarrow \max\{\mathcal{L}[0], \mathcal{L}[1], \ldots, \mathcal{L}[n-1]\}$;
    for $j \leftarrow k = \max\{lcp_{\max}, \ell_{\max}\}$ to 1 do
        foreach $i \in \mathcal{M}_{LCP}[j]$ do
            $\mathcal{UF}$.Union($i-1, i$);
        foreach $i \in \mathcal{M}_{\mathcal{L}}[j]$ do
            $f \leftarrow \mathcal{UF}$.Find(iSA[$i$]);
            Insert $i$ at the head of $\mathcal{A}[f]$;
    for $j \leftarrow 0$ to $n-1$ do
        foreach $i \in \mathcal{A}[j]$ do
            INSERT($i$, PSA);
    return PSA;
```

Example 7 (Running example). The following two tables show the partition of $\{0, 1, \ldots, n-1\}$ before (top) and after (bottom) the Union operations performed for $j = 1$. Each monochromatic block represents a set in the partition.

i	0	▨ 1	▨ 2	▨ 3	4	▨ 5	6	7	▨ 8
LCP[i]	0	1	2	3	1	0	1	2	0

i	0	1	2	3	4	▨ 5	▨ 6	▨ 7	8
LCP[i]	0	1	2	3	1	0	1	2	0
$\mathcal{L}[i]$	4	3	2	1	3	2	3	2	1

Find operations are then performed for those i for which $\mathcal{L}[i] = 1$. For example for $i = 3$ we have that Find(iSA[3]) = Find(7) = 5, since 5 is the smallest element in the set where 7 belongs. Hence 3 is added in the head of the linked list $\mathcal{A}[5]$.

Putting together Lemma 4, Theorem 6 and the above description we obtain the following.

Theorem 8. *Problem* PROPERTY SUFFIX ARRAY *can be solved in time and space $\mathcal{O}(n)$.*

In the standard setting, the SA is usually coupled with the LCP array to allow for efficient on-line pattern searches (see [20] for the details).

Definition 9. *The* property Longest Common Prefix array *(*pLCP*) for* x *and* $\mathcal{L}$ *is an integer array of size* n *such that, for all* $1 \leq r < n$, $\mathsf{pLCP}[r]$ *is the length of the longest common prefix of* $x[i\,..\,i+\mathcal{L}[i]-1]$ *and* $x[j\,..\,j+\mathcal{L}[j]-1]$*, where* $i = \mathsf{PSA}[r]$ *and* $j = \mathsf{PSA}[r-1]$*.*

Lemma 10. *We can compute the* pLCP *array in time* $\mathcal{O}(n)$*.*

Proof. We compute the pLCP array while constructing the PSA as follows. If we read both $\mathsf{PSA}[r-1]$ and $\mathsf{PSA}[r]$ from $\mathcal{A}[j]$, we set $\mathsf{pLCP}[r] = \mathcal{L}[\mathsf{PSA}[r-1]]$ since $x[i\,..\,i+\mathcal{L}[i]-1]]$ is a prefix of $x[i'\,..\,i'+\mathcal{L}[i']-1]]$. Otherwise, we read $\mathsf{PSA}[r-1]$ from $\mathcal{A}[j]$ and $\mathsf{PSA}[r] = i'$ from $\mathcal{A}[j']$ and proceed as follows:

1. If $\mathsf{iSA}[i'] < \mathsf{iSA}[i]$ then $x[i\,..\,i+\mathcal{L}[i]-1]$ is a prefix of $x[i'\,..\,i'+\mathcal{L}[i']-1]$ and hence we set $\mathsf{pLCP}[r] = \mathcal{L}[i]$;
2. Else $\mathsf{iSA}[i] < \mathsf{iSA}[i']$, and since $\mathcal{L}[i] \leq \mathsf{lcp}(j, \mathsf{iSA}[i])$ and $\mathcal{L}[i'] \leq \mathsf{lcp}(j', \mathsf{iSA}[i'])$ we set $\mathsf{pLCP}[r] = \min\{\mathsf{lcp}(j,j'), \mathcal{L}[i], \mathcal{L}[i']\}$.

We can compute $\mathsf{lcp}(j,j')$ for all consecutive non-empty lists $\mathcal{A}[j]$, $\mathcal{A}[j']$ in a simple scan of the LCP array in time $\mathcal{O}(n)$. □

Remark 11. Alternatively, we can compute the pLCP array using lcp queries, since $\mathsf{pLCP}[r] = \min\{\mathsf{lcp}(\mathsf{PSA}[r-1], \mathsf{PSA}[r]), \mathcal{L}[\mathsf{PSA}[r-1]], \mathcal{L}[\mathsf{PSA}[r]]\}$.

Finally, it is worth noting that the algorithms presented in this section for constructing the PSA depend neither on the fact that $\mathcal{L}[i] \geq \mathcal{L}[i-1]-1$ nor on the fact that we have (at most) one substring starting at each position. As a byproduct we thus obtain the following result *without* the aid of suffix tree, which is interesting in its own right.

Theorem 12. *Given* q *substrings of a string* x *of length* n*, encoded as intervals over* x*, we can sort them lexicographically in time* $\mathcal{O}(n+q)$*.*

4 Weighted Suffix Array

A *weighted sequence* X of length $|X| = n$ over an alphabet Σ is an $n \times \sigma$ matrix that specifies, for each position $i \in \{0, \dots, n-1\}$ and letter $c \in \Sigma$, a probability $\pi_i^{(X)}(c)$ of c occurring at position i. If the considered weighted sequence is unambiguous, we write π_i instead of $\pi_i^{(X)}$. These values are non-negative and sum up to 1 for any given i.

The *probability of matching* of a string p with a weighted sequence X ($|p| = |X|$) equals

$$\mathcal{P}(p, X) = \prod_{i=0}^{|p|-1} \pi_i^{(X)}(p[i]).$$

We say that a string p *matches* a weighted sequence X with probability at least $\frac{1}{z}$ if $\mathcal{P}(p, X) \geq \frac{1}{z}$. By $X[i..j]$ we denote a weighted sequence called a *factor* of X

and equal to $X[i] \dots X[j]$. We then say that a string p *occurs* in X at position i if p matches the factor $X[i \, .. \, i + |p| - 1]$.

A weighted sequence is called *special* if, at each position, it contains at most one letter with positive probability. In this special case the assumption that the probabilities sum up to 1 for a given position is waived.

In this section, we present an algorithm for constructing a new index for a weighted sequence X of length n and a probability threshold $\frac{1}{z}$. We combine the ideas presented above with the following powerful combinatorial result (Theorem 13) presented in [5]. Informally, Theorem 13 tells us that one can construct in $\mathcal{O}(nz)$ time a family of $\lfloor z \rfloor$ special weighted sequences, each of length n, that carry all the information about all the strings occurring in X. More specifically, a string occurs with probability $\beta \geq \frac{1}{z}$ at position i in one of these $\lfloor z \rfloor$ special weighted sequences if and only if it occurs at position i of X with probability β. The authors of [5] used this result to design an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space algorithm for constructing the *Weighted Index*: an $\mathcal{O}(nz)$-sized suffix-tree-like index for X. The Weighted Index is essentially the PST built over this family of strings after some appropriate property shifting.

Theorem 13 ([5]). *For a weighted sequence X of length n over an integer alphabet of size σ and a threshold $\frac{1}{z}$, one can construct in $\mathcal{O}(n\sigma + nz)$ time an equivalent collection of $\lfloor z \rfloor$ special weighted sequences.*

Definition 14. *The* Weighted Suffix Array *(WSA) for X and $\frac{1}{z}$ is an integer array (of size at most $n\lfloor z \rfloor$) storing the path-labels of the terminal nodes of the Weighted Index for X and $\frac{1}{z}$ in the order in which they are visited in a (lexicographic) depth first traversal.*

The idea is to create a new special weighted sequence Y by concatenating these $\lfloor z \rfloor$ special weighted sequences. At this point we view Y as the standard string y of length $n\lfloor z \rfloor$ (at most one letter per position has a positive probability). The probabilities at each position of Y and the ends of the original $\lfloor z \rfloor$ special weighted sequences give array $\mathcal{L}$ for y. We then construct the PSA for y and $\mathcal{L}$.

We are not done yet since a string of length m occurring at a position i of X may occur at several positions $j_0, j_1, \dots, j_{k-1}$ in y, with $j_p = i \pmod n$ and $\mathcal{L}[j_p] = m$ for all $0 \leq p < k$. We naturally want to keep *one* of these occurrences. We do that as follows: we identify maximal intervals $[r, s]$ in the PSA satisfying $\mathcal{L}[\mathsf{PSA}[q]] = \mathsf{pLCP}[t] = m$ for all $r - 1 \leq q \leq s$ and $r \leq t \leq s$; for each such interval, we consider all of the indices in $\{\mathsf{PSA}[q] | r - 1 \leq q \leq s\}$ modulo n, we bucket sort the residuals, and finally keep one representative for each of them. Doing this for the PSA of y and $\mathcal{L}$ from left to right, we end up with an array of size *at most* $n\lfloor z \rfloor$ that is the WSA for X and $\frac{1}{z}$.

Theorem 15. *The WSA for a weighted sequence X of length n over an integer alphabet of size σ and a threshold $\frac{1}{z}$ can be constructed in $\mathcal{O}(n\sigma + nz)$ time.*

The WSA for X and $\frac{1}{z}$, coupled with the naturally defined *weighted Longest Common Prefix array* (wLCP), which can be inferred directly from the pLCP array of y and $\mathcal{L}$, is an index with comparable capabilities as the ones of the SA coupled with the LCP array in the standard setting [20].

Example 16. Let $X = [(\mathtt{a}, 0.5), (\mathtt{b}, 0.5)]\mathtt{bab}[(\mathtt{a}, 0.5), (\mathtt{b}, 0.5)][(\mathtt{a}, 0.5), (\mathtt{b}, 0.5)]$ and $\frac{1}{z} = 1/4$. The family of z strings and the corresponding index are as follows:

i	0	1	2	3	4	5
	a	b	a	b	b	b
	a	b	a	b	a	b
	b	b	a	b	b	a
	b	b	a	b	a	a

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$y[i]$	a	b	a	b	b	b	a	b	a	b	a	b	b	b	a	b	b	a	b	b	a	b	a	a
WSA$[i]$	17	22	10	20	8	6	0	14	2	5	16	21	9	19	7	13	1	4	15	18	12	3		
$\mathcal{L}$[WSA$[i]$]	1	2	2	4	4	5	5	4	4	1	2	3	3	5	5	5	5	2	3	5	5	3		
wLCP$[i]$	0	1	1	2	3	4	4	2	3	0	1	2	2	3	4	3	4	1	2	3	4	2		

5 Experimental Results

We have implemented algorithms ST-PSA and UF-PSA to compute the PSA. The programs have been implemented in the C++ programming language and developed under the GNU/Linux operating system. The input parameters for both programs are a string of length n and an integer array of size n for the corresponding Π-valid intervals. The output of both programs is the PSA. The source code is distributed at https://github.com/YagaoLiu/WSA under the GNU General Public License. For comparison purposes, we used the implementation of the PST from [6] which has a similar interface (https://bitbucket.org/kociumaka/weighted_index). All experiments have been conducted on a Desktop PC using one core of Intel Xeon CPU E5-2640 at 2.60 GHz. All programs have been compiled with g++ version 6.2.0 at optimization level 3 (-O3).

It is well-known, among practitioners and elsewhere, that optimal RMQ data structures for on-line $\mathcal{O}(1)$-time querying carry high constants in their preprocessing and querying time [3]. One would not thus expect that algorithm LCP-PSA performs well in practice. Indeed, we have implemented LCP-PSA but we omit its presentation here since it was too slow for the same inputs.

To evaluate the time and space performance of our implementations, we used synthetic weighted DNA sequences ($\sigma = 4$). We used the weighted sequences to create a new standard string y and compute the integer array $\mathcal{L}$ as described in Sect. 4. Thus given a weighted sequence of length n and a probability threshold $\frac{1}{z}$, we obtained a new string of length nz. In our experiments, we used weighted sequences of length ranging from 125,000 to 4,000,000; the probability threshold was set to 1/8. The strings obtained from weighted sequences are thus of length ranging from 1,000,000 to 32,000,000. The results are plotted in Figs. 1 and 2. In Fig. 1 we see that: (i) UF-PSA and PST run in *linear* time; (ii) ST-PSA runs in (slightly) *super-linear* time; and (iii) UF-PSA is the *fastest* of the three implementations. In Fig. 2 we see that: (i) all three implementations run in *linear* space; (ii) PST is by far the most space *inefficient* of the three implementations; and (iii) ST-PSA is the most space *efficient* of the three implementations. The presented experimental results confirm *fully* our theoretical findings and justify the motivation for the contributions of this paper.

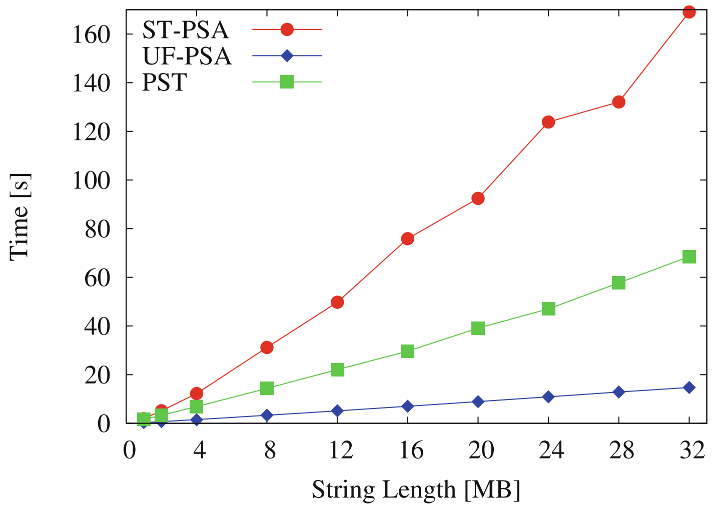


Fig. 1. Elapsed time of ST-PSA, UF-PSA, and PST using synthetic texts of length ranging from 1 MB to 32 MB over the DNA alphabet.

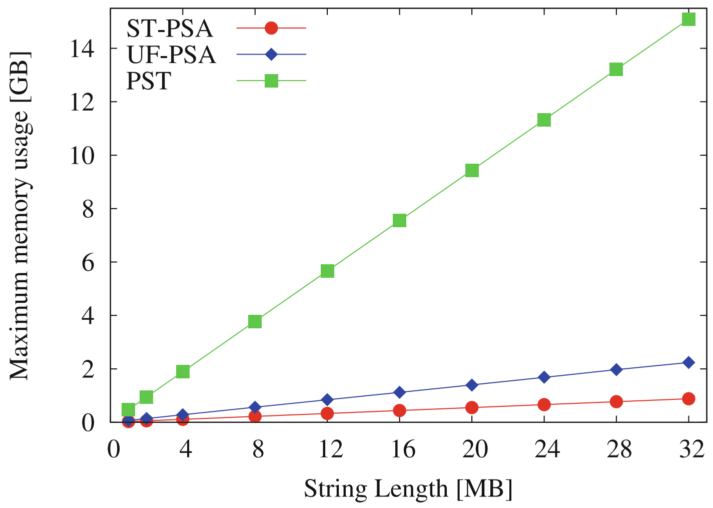


Fig. 2. Peak memory usage of ST-PSA, UF-PSA, and PST using synthetic texts of length ranging from 1 MB to 32 MB over the DNA alphabet.

References

1. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. J. Discrete Algorithms **2**(1), 53–86 (2004)
2. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. IEEE Trans. Knowl. Data Eng. **21**(5), 609–623 (2009)

3. Alzamel, M., Charalampopoulos, P., Iliopoulos, C.S., Pissis, S.P.: How to answer a small batch of RMQs or LCA queries in practice. In: IWOCA. LNCS. Springer International Publishing (2017, in press)
4. Amir, A., Chencinski, E., Iliopoulos, C., Kopelowitz, T., Zhang, H.: Property matching and weighted matching. Theor. Comput. Sci. **395**(2–3), 298–310 (2008)
5. Barton, C., Kociumaka, T., Liu, C., Pissis, S.P., Radoszewski, J.: Indexing weighted sequences: neat and efficient. CoRR abs/1704.07625v1 (2017)
6. Barton, C., Kociumaka, T., Liu, C., Pissis, S.P., Radoszewski, J.: Indexing weighted sequences: neat and efficient. CoRR abs/1704.07625v2 (2017)
7. Barton, C., Kociumaka, T., Pissis, S.P., Radoszewski, J.: Efficient index for weighted sequences. In: CPM. LIPIcs, vol. 54, pp. 4:1–4:13. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
8. Barton, C., Liu, C., Pissis, S.P.: On-line pattern matching on uncertain sequences and applications. In: Chan, T.-H.H., Li, M., Wang, L. (eds.) COCOA 2016. LNCS, vol. 10043, pp. 547–562. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48749-6_40
9. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000). https://doi.org/10.1007/10719839_9
10. Biswas, S., Patil, M., Thankachan, S.V., Shah, R.: Probabilistic threshold indexing for uncertain strings. In: EDBT. pp. 401–412 (2016). OpenProceedings.org
11. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education, Pennsylvania (2001)
12. Crochemore, M., Iliopoulos, C., Kubica, M., Radoszewski, J., Rytter, W., Stencel, K., Walen, T.: New simple efficient algorithms computing powers and runs in strings. Discrete Appl. Math. **163**(Part 3), 258–267 (2014)
13. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. J. Comput. Syst. Sci. **30**(2), 209–221 (1985)
14. Iliopoulos, C.S., Rahman, M.S.: Faster index for property matching. Inf. Process. Lett. **105**(6), 218–223 (2008)
15. Juan, M.T., Liu, J.J., Wang, Y.L.: Errata for "faster index for property matching". Inf. Process. Lett. **109**(18), 1027–1029 (2009)
16. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A. (ed.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48194-X_17
17. Kociumaka, T., Pissis, S.P., Radoszewski, J.: Pattern matching and consensus problems on weighted sequences and profiles. In: ISAAC. LIPIcs, vol. 64, pp. 46:1–46:12. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
18. Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Walen, T.: Efficient algorithms for shortest partial seeds in words. Theor. Comput. Sci. **710**, 139–147 (2018)
19. Kopelowitz, T.: The property suffix tree with dynamic properties. Theor. Comput. Sci. **638**(C), 44–51 (2016)
20. Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
21. Nong, G., Zhang, S., Chan, W.H.: Linear suffix array construction by almost pure induced-sorting. In: DCC, pp. 193–202. IEEE (2009)
22. Weiner, P.: Linear pattern matching algorithms. In: SWAT (FOCS), pp. 1–11. IEEE Computer Society (1973)

Competitive Algorithms for Demand Response Management in Smart Grid

Vincent Chau[1], Shengzhong Feng[1], and Nguyen Kim Thang[2](✉)

[1] Shenzhen Institutes of Advanced Technology, Academy of Sciences, Shenzhen, China
{vincentchau,sz.feng}@siat.ac.cn

[2] IBISC, Univ Évry, Université Paris-Saclay, 91025 Évry, France
thang@ibisc.fr

Abstract. We consider a scheduling problem which abstracts a model of demand-response management in Smart Grid. In the problem, there is a set of unrelated machines and each job j (representing a client demand) is characterized by a release date, and a power request function representing its request demand at specific times. Each machine has an energy power function and the energy cost incurred at a time depends on the load of the machine at that time. The goal is to find a non-migration schedule that minimizes the total energy (over all times).

We give a competitive algorithm for the problem in the online setting where the competitive ratio depends (only) on the power functions of machines. In the setting with typical energy function $P(z) = z^{\nu}$, the algorithm is $\Theta(\nu^{\nu})$-competitive, which is *optimal* up to a constant factor. Our algorithm is *robust* in the sense that the guarantee holds for arbitrary request demands of clients. This enables flexibility on the choices of clients in shaping their demands — a desired property in Smart Grid.

We also consider a special case in offline setting in which jobs have unit processing time, constant power request and identical machines with energy function $P(z) = z^{\nu}$. We present a 2^{ν}-approximation algorithm for this case.

1 Introduction

Electrical Smart Grid is one of the major challenges in the 21st century [20]. It is a network of electricity distribution that promotes the traffic information between producers and consumers in order to adjust the electricity flow in real time, i.e. it aims to improve the journey of the electricity through information and communication technologies in contrast of the traditional power system. It has been raised in [6] that in the US power grid, 10% of all generation assets

V. Chau and S. Feng—Research supported by NSFC (no. 61433012, U1435215), by Shenzhen basic research grant JCYJ20160229195940462 and by National High-Tech R&D Program of China (863 Program) 2015AA050201.

N. K. Thang—Research supported by the ANR project OATA no. ANR-15-CE40-0015-01, Hadamard PGMO and DIM RFSI.

and 25% of distribution infrastructure are required for less than 400 hours per year, which represents roughly 5% of the time [20]. A smart grid is a power grid system that optimizes the efficiency of the power generation, distribution and consumption, and eventually the storage, of the energy in order to coordinate the electric network, from the production to the consumer. It can be noticed that the power grid may not be efficient during the peak demand hour if the management of the smart grid is not well handled. Indeed, the cost of the electricity production can be high if there is a high demand, and the electricity suppliers may charge the consumer according to the generation cost. Therefore, the cost of the electricity can be different over time, intuitively we have a lower price during off-peak hours and a higher price during peak hours. That is why *demand response management* [10] has been studied in order to overcome this problem. The goal of each user is to minimize his own cost by requesting the electricity during off-peak hours and reduce the peak load while satisfying his demand. Thus, *demand response management* is essentially beneficial to the consumers.

In fact, it can be seen as a scheduling problem. Each user is a job with the same release date and deadline in which the job cannot be schedule before the release date, nor after the deadline. Furthermore, a user is defined by an electricity demand over time which can also be represented in the scheduling problem. Finally, we have a cost that will be charged to users depending on the load at each moment. The goal is to minimize the total cost while satisfying all demands. A more formal definition is given in the next section.

In this paper, we consider a general online scheduling problem which models the demand response management and we design algorithms towards the following main purposes of Smart Grid:

– Optimizing the energy consumption.
– Enabling customer choice and letting them react rationally.

1.1 Model Definition

We consider the following scheduling problem. We are given m unrelated machines and a set of n jobs. Here, machines represents different resources in a smart grid or different electrical sub-networks. Each job represents the demand of a client and the demand is customized by the client. Specifically, each job j is characterized by its release date r_j and an *arbitrary* power request function $h_{i,j,k} : \mathbb{N} \rightarrow \mathbb{R}^+$, meaning that if a job j starts at time k in machine i then its request demand at time t is $h_{i,j,k}(t)$. We denote the *execution* of job j as $s_{i,j,k}$ if job j is processed in machine i with the starting time k. In the problem, *migration* of jobs between machines is not allowed. (In other words, a job must be executed in exactly one machine.) It is a desired property in various systems since in case of migration, the communications and storage/reloading data of jobs are costly.

Given a schedule (executions of all jobs), the total *load* at time t in machine i is $\sum_{j,k} h_{i,j,k}(t)$ where the sum is taken over all job executions in machine i. The total *energy* is defined as $\sum_i \sum_t P_i\big(\sum_{j,k} h_{i,j,k}(t)\big)$ where P_i is an energy power function of machine i. Typically, $P_i(z) = z^{\nu_i}$ for some constant parameter $\nu_i \geq 1$. In the problem, we consider P_i as *arbitrary* non-decreasing functions, and possibly non-convex. The goal is find a feasible schedule that minimize the total energy consumption over all times.

In the paper, we consider both offline and online settings. In the offline setting, the scheduler has the full knowledge on all parameters while in the online setting, jobs arrive over time and the scheduler is aware of jobs (and their parameters) only at their arrival time. The online setting is appropriate to the dynamic nature of the demand response management.

The presented model encompasses the previous ones [11] in literature. In the latter, jobs have release date r_j, deadline d_j, processing time $p_{i,j}$ and jobs have to be processed non-preemptively. The power request of a job j is some constant h_j during its non-preemptive execution. It is captured by the model by defining

$$h_{i,j,k}(t) = \begin{cases} h_j & \text{if } r_j \leq k \leq d_j - p_{ij} \text{ and } t \in [k, k+p_{ij}], \\ 0 & \text{if } r_j \leq k \leq d_j - p_{ij} \text{ and } t \notin [k, k+p_{ij}], \\ \infty & \text{otherwise.} \end{cases} \tag{1}$$

Geometrically, in the model as shown in Eq. (1), each job corresponds to a rectangle and the problem essentially consists of packing rectangles to minimize the total energy. In this case, the power request is constant from the beginning to the end of the request. For short, we call the model defined by Eq. (1) *rectangle scheduling*. In our model, there is no condition on the demand (i.e., energy request) of jobs and the demands can be specifically customized by clients. Geometrically, each job in our model has an *arbitrary* (not necessarily continuous) form which represents varying power requests during its execution (See Fig. 1). Hence, the model offers flexible choices to clients along the line of Smart Grid's purposes.

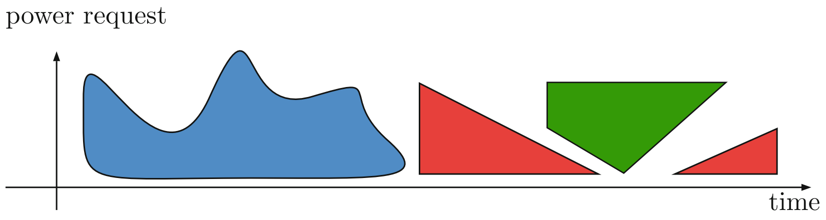


Fig. 1. Example of a schedule with arbitrary power requests during execution of jobs.

1.2 Related Works

In this section, we summarize related works in the model of *rectangle scheduling* which, to the best of our knowledge, is the only one that has been studied so far.

Koutsopoulos and Tassiulas [11] formulated the rectangle scheduling model where the cost function is piecewise linear. They show that the problem is NP-hard, and it can further be adapted to show the NP-hardness of the general problem where the cost function is convex [5]. In the offline setting, Burcea et al. [5] gave a polynomial time algorithms for the case of unit height (i.e., unit power request) and unit width (i.e., duration of request). Furthermore, in the full version of [13] (see [14]), Liu et al. showed that the offline case, where jobs have unit processing time but with arbitrary power request, admits a $2^{\nu+1}$-approximation algorithm which is based on the results of the the dynamic speed scaling problem [1,4,21].

In the online setting, [9] proposed a simple greedy algorithm which is 2-competitive for the unit case and the power function is z^2. However, [13] showed that the greedy algorithm is in fact at least 3-competitive by providing a counter example. In the same paper, Liu et al. [13] considered the single machine setting in which they presented an online $\Theta\left(\log^\nu\left(\frac{p_{\max}}{p_{\min}}\right)\right)$-competitive algorithm where $p_{\min} = \min_j\{p_j : p_j > 0\}$ and $p_{\max} = \max_j p_j$. This is the best known algorithm (even in offline setting) where jobs have arbitrary width, arbitrary height and the power energy function is z^ν. Furthermore, for special cases of jobs with unit processing time, Liu et al. [13] also gave competitive algorithms. A summary of the results can be found in Table 1.

Table 1. Summary of competitive ratios in the rectangle scheduling model with power function z^ν for a *single* machine. Our result holds for *unrelated machines*.

Processing time p_j	Power request h_j	Prior best-known results	Our result
Unit	Uniform (i.e., $h_j = h_{j'}$)	$\min\{(4\nu)^\nu/2 + 1, 2^\nu(8e^\nu + 1)\}$ [13]	
	Arbitrary	$2^\nu(8e^\nu + 1)$ [13]	
Arbitrary	Arbitrary	$O\left(\log^\nu\left(\frac{p_{\max}}{p_{\min}}\right)\right)$ [13] $\Omega(\nu^\nu)$ [13]	$\boldsymbol{\Theta}(\nu^\nu)$

Besides, Salinas et al. [18] considered a multi-objective problem to minimize energy consumption cost and maximize some utility that can be the profit for the operator as well as for the clients. On the other hand, a related problem is to manage the load by considering different price of electricity over time [8,16]. Recent surveys of the area can be found in [2,10,15].

1.3 Our Contribution and Approaches

In this paper, we investigate the online and offline aspects of the problem.

Online Setting. The main result of the paper is a competitive algorithm for the problem in online setting where the competitive ratio is characterized by a notion called *smoothness* [17,19] of the machine energy power functions. Informally, the algorithm assigns and executes each job that arrives on a machine in such a way that minimizes the marginal increase of the total cost.

In designing a competitive algorithm for the problem, we consider a primal-dual approach. The main difficulty in proving the performance of the algorithm is that all known LPs has unbounded integrality gap, even for the special case of rectangle scheduling. Intuitively, the drawback of all known LPs is that in the optimal fractional solution, jobs are fractionally assigned to machines while in the optimal integer solution, migration of jobs is not allowed. To bypass this obstacle, we consider the primal-dual framework based on configuration linear programs in [19]. The framework is presented in order to reduce the integrality gap and also to study problems with non-linear, non-convex objective functions. The approach is particularly useful since the energy power functions are non-linear. Employing the techniques from [19], we derive a greedy algorithm with competitive ratio characterized by the notion of *smoothness*, which is defined as follows.

Definition 1. *A function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is* (λ, μ)-smooth *if for any sets of non-negative numbers $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, the following inequality holds:*

$$\sum_{i=1}^{n} \left[f\Bigl(a_i + \sum_{j=1}^{i} b_j\Bigr) - f\Bigl(\sum_{j=1}^{i} b_j\Bigr) \right] \leq \lambda \cdot f\Bigl(\sum_{i=1}^{n} a_i\Bigr) + \mu \cdot f\Bigl(\sum_{i=1}^{n} b_i\Bigr)$$

A set of cost functions $\{f_e : e \in \mathcal{E}\}$ is (λ, μ)-smooth *if every function f_e is* (λ, μ)-smooth.

Specifically, in the problem, assuming that all energy power functions are (λ, μ)-*smooth* for some $\lambda > 0$ and $0 < \mu < 1$, our algorithm is $\lambda/(1-\mu)$-competitive. For energy power functions of forms $P_i(z) = z^{\nu_i}$, they are $\bigl(O(\nu^{\nu-1}), \frac{\nu-1}{\nu}\bigr)$-smooth where $\nu = \max_i \nu_i$. That leads to the competitive ratio $O(\nu^\nu)$ which improves upon the best-known $\Theta\left(\log^\nu\left(\frac{p_{\max}}{p_{\min}}\right)\right)$-competitive algorithm where $p_{\min} = \min_j\{p_j : p_j > 0\}$ and $p_{\max} = \max_j p_j$. Our competitive ratio is not only independent on the jobs' parameters but it is indeed *optimal* up to a constant factor. The matching lower bound is given by [13, Theorem 9] for a single machine in the rectangle scheduling model. In particular, [13] gave a lower bound which is $\frac{1}{3}\bigl(\log \frac{p_{\max}}{p_{\min}}\bigr)^\nu$ where $\log \frac{p_{\max}}{p_{\min}} = \nu$.

Our greedy algorithm has several interesting features toward the purposes of Smart Grid. First, the algorithm is simple and easy to implement which makes it practically appealing. Note that despite the simplicity of our algorithm, no bounded competitive ratio has been known even for the rectangle scheduling model. Second, the algorithm performance guarantee holds for jobs with arbitrary varying power requests (arbitrary forms). Apart of answering open questions raised in [13], it is particularly useful for the demand response management.

Once the algorithm is publicly given and clients are charged accordingly to the marginal increase of the total energy cost, clients can *arbitrarily* customized their demand in order to minimize their payment. This property is desirable since it enables the clients to react rationally. In the side of the smart grid management, no modification in the algorithm is needed while always maintaining the competitiveness (optimality in case of typical energy functions).

Offline Setting. In offline setting, we give an improved 2^{ν}-approximation algorithm when jobs have unit processing time. This result improves upon the $2^{\nu+1}$-approximation algorithm given by Liu et al. [14] in two aspects. First, it slightly improves the competitive ratio. Secondly, our result holds for multiple (identical) machine environment. Our algorithm makes use of the approximation algorithm for scheduling problems with convex norm objective functions given by [3]. The latter is designed by solving a convex relaxation and round to an integer solution using the Lenstra-Shmoys-Tardos scheme [12].

2 A Competitive Online Algorithms

Formulation. In the model, the execution of a job is specified by two parameters: (1) a machine in which it is executed; and (2) a starting time. Note that these parameters fully represent the demand of a job, including the power request at any time t during its execution. Formally, we denote the *execution* of job j as $s_{i,j,k}$ if job j is processed in machine i with the starting time k. Recall that if the execution of a job j is $s_{i,j,k}$ then the request demand of the job at time t is $h_{i,j,k}(t)$. Let $\mathcal{S}_j$ be a set of feasible executions of job j. For example, in the rectangle scheduling model, $\mathcal{S}_j$ consists of $s_{i,j,k}$ for all machines i and starting time k such that $r_j \leq k \leq d_j - p_{ij}$. As the set of machines and times[1] are finite, so is the set $\mathcal{S}_j$ for every job j. Let $x_{i,j,k}$ be a variable indicating whether the execution of job j is $s_{i,j,k} \in \mathcal{S}_j$. We say that A is a *scheduling configuration* (configuration in short) in machine i if A is a feasible schedule of a subset of jobs. Specifically, A consists of tuples (i, j, k) meaning that the execution of job j is $s_{i,j,k}$. For a scheduling configuration A and machine i, let $z_{i,A}$ be a variable such that $z_{i,A} = 1$ if and only if for every tuple $(i, j, k) \in A$, we have $x_{i,j,k} = 1$. In other words, $z_{i,A} = 1$ if and only if the schedule in machine i follows *exactly* the configuration A. Given a scheduling configuration A, let $A(t)$ be the load (height) of the corresponding schedule at time t. We denote the energy cost of a configuration A of machine i as $c_i(A) := \sum_t P_i(A(t))$. We consider the following formulation and the dual of its relaxation.

[1] For convenience, we consider schedules up to a time T, which can be arbitrarily large but finite.

$$\min \sum_{i,A} c_i(A) z_{i,A}$$

$$\sum_{i,k:s_{i,j,k}\in\mathcal{S}_j} x_{i,j,k} = 1 \qquad \forall j$$

$$\sum_{A:(i,j,k)\in A} z_{i,A} = x_{i,j,k} \qquad \forall i,j,k$$

$$\sum_{A} z_{i,A} = 1 \qquad \forall i$$

$$x_{i,j,k}, z_{i,A} \in \{0,1\} \qquad \forall i,j,k,A$$

$$\max \sum_j \alpha_j + \sum_i \gamma_i$$

$$\alpha_j \le \beta_{i,j,k} \qquad \forall i,j,k$$

$$\gamma_i + \sum_{(i,j,k)\in A} \beta_{i,j,k} \le c_i(A) \qquad \forall i,A$$

In the primal, the first constraint guarantees that a job j has to be processed by some feasible execution (in some machine). The second constraint ensures that if job j follows the execution $s_{i,j,k}$ then in the solution, the scheduling configuration of machine i must contain the tuple (i,j,k) corresponding to execution $s_{i,j,k}$. The third constraint says that in the solution, there is always a scheduling configuration (possibly empty set) associated to machine i.

Algorithm. We first interpret intuitively the dual variables, dual constraints and derive useful observations for a competitive algorithm. Variable α_j represents the increase of energy to the arrival of job j. Variable $\beta_{i,j,k}$ stands for the marginal energy if job j follows execution $s_{i,j,k}$. By this interpretation, the first dual constraint clearly indicates the greedy behavior of an algorithm. That is, if a new job j is released, select an execution $s_{i,j,k} \in \mathcal{S}_j$ that minimizes the marginal increase of the total energy.

Formally, let A_i^* be the set of current schedule of machine i and initially, $A_i^* \leftarrow \emptyset$ for every machine i. At the arrival of job j, select an execution $s_{i^*,j,k^*} \in \mathcal{S}_j$ such that

$$s_{i^*,j,k^*} \in \arg\min_{s_{i,j,k}\in\mathcal{S}_j} \left[c_i(A_i^* \cup s_{i,j,k}) - c_i(A_i^*)\right]$$

or equivalently,

$$s_{i^*,j,k^*} \in \arg\min_{s_{i,j,k}\in\mathcal{S}_j} \sum_t \left[P_i\Big(A_i^*(t) + h_{i,j,k}(t)\Big) - P_i\Big(A_i^*(t)\Big)\right]$$

where $(A_i^* \cup s_{i,j,k})$ is the current schedule with additional execution $s_{i,j,k}$ of job j. Note that in configuration $(A_i^* \cup s_{i,j,k})$, the load at time t in machine i is $P_i\big(A_i^*(t) + h_{i,j,k}(t)\big)$. Then assign job j to machine i^* and process it according to the corresponding execution of s_{i^*,j,k^*}.

Dual variables. Assume that all energy power functions P_i are (λ,μ)-smooth for some fixed parameters $\lambda > 0$ and $\mu < 1$, then we construct a dual feasible solution in the following way. Let $A_{i,\prec j}^*$ be the scheduling configuration of machine i (due to the algorithm) prior to the arrival of job j. Define α_j as $1/\lambda$ times the increase of the total cost due to the arrival of job j. In other words, if the algorithm selects the execution s_{i^*,j,k^*} for job j then

$$\begin{aligned}\alpha_j &= \frac{1}{\lambda}\Big[c_{i^*}(A^*_{i^*,\prec j} \cup s_{i^*,j,k^*}) - c_{i^*}(A^*_{i^*,\prec j})\Big] \\ &= \frac{1}{\lambda}\sum_t \Big[P_{i^*}\Big(A^*_{i^*,\prec j}(t) + h_{i^*,j,k^*}(t)\Big) - P_{i^*}\Big(A^*_{i^*,\prec j}(t)\Big)\Big]\end{aligned}$$

For each machine i and job j, we set

$$\begin{aligned}\beta_{i,j,k} &= \frac{1}{\lambda}\Big[c_i(A^*_{i,\prec j} \cup s_{i,j,k}) - c_i(A^*_{i,\prec j})\Big] \\ &= \frac{1}{\lambda}\sum_t \Big[P_i\Big(A^*_{i,\prec j}(t) + h_{i,j,k}(t)\Big) - P_i\Big(A^*_{i,\prec j}(t)\Big)\Big].\end{aligned}$$

Finally, for every machine i, we define the dual variable

$$\gamma_i = -\frac{\mu}{\lambda}c_i(A^*_i)$$

where A^*_i is the schedule on machine i (at the end of the instance).

Lemma 1. *The dual variables defined as above are feasible.*

Proof. By the definition of dual variables, the first constraint reads

$$\frac{1}{\lambda}\Big[c_{i^*}(A^*_{i^*,\prec j} \cup s_{i^*,j,k^*}) - c_{i^*}(A^*_{i^*,\prec j})\Big] \leq \frac{1}{\lambda}\Big[c_i(A^*_{i,\prec j} \cup s_{i,j,k}) - c_i(A^*_{i,\prec j})\Big]$$

This inequality follows immediately the choice of the algorithm.

We now show that the second constraint holds. Fix a machine i and an arbitrary configuration A on machine i. The corresponding constraint reads

$$\begin{aligned} & -\frac{\mu}{\lambda}c_i(A^*_i) + \frac{1}{\lambda}\sum_{(i,j,k)\in A}\Big[c_i(A^*_{i,\prec j} \cup s_{i,j,k}) - c_i(A^*_{i,\prec j})\Big] \leq c_i(A) \\ \Leftrightarrow\quad & \sum_{(i,j,k)\in A}\Big[c_i(A^*_{i,\prec j} \cup s_{i,j,k}) - c_i(A^*_{i,\prec j})\Big] \leq \lambda c_i(A) + \mu c_i(A^*_i) \\ \Leftrightarrow\quad & \sum_{(i,j,k)\in A}\sum_t \Big[P_i(A^*_{i,\prec j}(t) + h_{i,j,k}(t)) - P_i(A^*_{i,\prec j}(t))\Big] \\ & \qquad \leq \lambda \sum_t P_i(A(t)) + \mu \sum_t P_i(A^*_i(t)) \qquad (2)\end{aligned}$$

where $A^*_{i,\prec j}(t)$ is the load (height) of machine i (due to the algorithm) at time t before the arrival of job j.

Observe that $A^*_{i,\prec j}(t)$ is the sum of power requests (according to the algorithm) at time t of jobs assigned to machine i prior to job j. As the power function P_i is (λ, μ)-smooth, for any time t we have

$$\sum_{(i,j,k)\in A} \Big[P_i\big(A^*_{i,\prec j}(t) + h_{i,j,k}(t)\big) - P_i\big(A^*_{i,\prec j}(t)\big) \Big] \\ \leq \lambda P_i\Big(\sum_{(i,j,k)\in A} h_{i,j,k}(t) \Big) + \mu P_i\big(A^*_i(t)\big)$$

Summing over all times t, Inequality (2) holds. Therefore, the lemma follows.

We are now ready to prove the main theorem.

Theorem 1. *If all energy power functions are (λ, μ)-smooth, then the algorithm is $\lambda/(1-\mu)$-competitive. In particular, if $P_i(z) = z^{\nu_i}$ for $\nu_i \geq 1$ then the algorithm is $O(\nu^\nu)$-competitive where $\nu = \max_i \nu_i$.*

Proof. By the definitions of dual variables, the dual objective is

$$\sum_j \alpha_j + \sum_i \gamma_i = \sum_i \frac{1}{\lambda} c_i(A^*_i) - \sum_i \frac{\mu}{\lambda} c_i(A^*_i) = \frac{1-\mu}{\lambda} \sum_i c_i(A^*_i)$$

Besides, the cost of the solution due to the algorithm is $\sum_i c_i(A^*_i)$. Hence, the competitive ratio is at most $\lambda/(1-\mu)$.

Particularly, energy power functions of forms $P_i(z) = z^{\nu_i}$ for $\nu_i \geq 1$ are $\big(O(\nu^{\nu-1}), \frac{\nu-1}{\nu}\big)$-smooth for $\nu = \max_i \nu_i$. In fact, the smoothness follows (smooth) inequalities in [7], which states: for $\nu > 1$ and for any sets of non-negative numbers $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, it always holds that

$$\sum_{i=1}^n \left[\left(a_i + \sum_{j=1}^i b_j \right)^\nu - \left(\sum_{j=1}^i b_j \right)^\nu \right] \leq O(\nu^{\nu-1}) \cdot \left(\sum_{i=1}^n a_i \right)^\nu + \frac{\nu-1}{\nu} \cdot \left(\sum_{i=1}^n b_i \right)^\nu$$

That implies the competitive ratio ν^ν of the algorithm for power functions $P_i(z) = z^{\nu_i}$. □

3 An Approximation Algorithm for Unit Processing Time Jobs

In this section, we investigate the offline case in identical machine environment where jobs have unit processing time but different power requests on different machines. Note that this corresponds to the restricted model of *rectangle scheduling*. We consider typical energy power function $P(z) = z^\nu$ for every machine and we assume that jobs need to be assigned to time-slot. This problem is proved to be NP-hard by a reduction to the 3-Partition problem even for the case where jobs have common release time and common deadline [5].

Let $\Theta = \cup_{j=1}^{n}\{r_j + a \mid a = -n, \ldots, n\} \cup_{j=1}^{n} \{d_j + a \mid a = -n, \ldots, n\}$ to be a set of time-slots. We show that it is sufficient to consider only schedules in which jobs are processed within these time-slots. In particular, this set contains $O(n^2)$ time-slots and will help to design a polynomial time approximation algorithm.

Lemma 2. *The schedules in which jobs start at a date in Θ are* dominant*. In other words, any schedule can be transformed to one in which jobs start at a date in Θ without increasing the cost.*

Proof. It is sufficient to consider a machine and show how to transform the schedule of the machine to the new one such that each job starts at a date in Θ without increasing the cost.

Let t be the first moment where jobs that are assigned to this time-slot does not belong to Θ. We consider the maximal continuous interval from time-slot t in which every time-slot has at least one job that is assigned. If the considered interval is $[t, u)$, then the time-slot $u + 1$ is idle.

First, we observe that the length of this interval is lower or equal to n. Indeed, in the worst case, each job is assigned to different time-slot. We shift this interval, as well as the jobs, by one unit time to the right, i.e. after the shift, the interval will be $[t + 1, u + 1)$.

Three cases may occur (see Fig. 2):

- We reach another job. We then consider the new maximal continuous interval and continue to shift it.
- We reach a deadline. The starting time of the interval must be in Θ since the length of the interval is at most n.
- None of the above cases, we continue to shift the interval to the right.

By this operation, we observe that the cost of the schedule remains the same because the costs of time-slots are independent. By doing a such modification, jobs are executed at the same way, with the same order and with the same cost, the only difference is the time-slots in which the jobs are executed. □

The main idea is to reduce the smart grid problem to the following L_ν*-norm problem.* In the latter, we are given a set $\mathcal{J}$ of n jobs and a set $\mathcal{M}$ of m unrelated machine. Each job $j \in \mathcal{J}$ have a processing time $p_{i,j}$ if it is assigned to machine i. We define the decision variable $y_{i,j} = 1$ if the job j is assigned to machine i, and $y_{i,j} = 0$ otherwise. The goal is to minimize the following function:

$$\sqrt[\nu]{\sum_{i \in \mathcal{M}} \left(\sum_{j=1}^{n} y_{i,j} p_{i,j} \right)^{\nu}} \tag{3}$$

Lemma 3. *The problem of smart grid with unit processing time jobs and identical machines can be polynomially reduced to the L_ν-norm minimization problem on unrelated machines.*

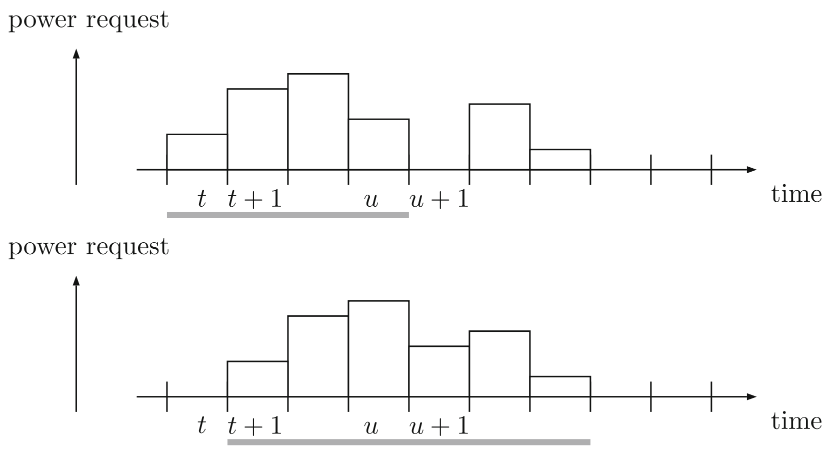


Fig. 2. Illustration of a shift of an interval. After the shift, the former interval meet another job. We then need to consider the continuous interval from time-slot $t + 1$. It corresponds to the first case in the proof of Lemma 2.

Proof. By Lemma 2, there is a polynomial number of time-slots to which jobs can be assigned. We create a corresponding machine (i, t) for each time-slot $t \in \Theta$ and each machine i. Similarly, we create a new job $j' \in \mathcal{J}$ (in L_ν-norm problem) which corresponds to job $j \in J$ (in the smart grid problem) in the following way:

$$p_{(i,t),j'} = \begin{cases} h_j & \text{if } t \in [r_j, d_j) \\ +\infty & \text{otherwise} \end{cases} \tag{4}$$

Given a schedule for the L_ν-norm problem with cost C, we show how to build a feasible schedule for the smart grid problem with cost C^ν.

For each job $j \in \mathcal{J}$ that is assigned to machine $(i, t) \in \mathcal{M}$ in the L_ν-norm problem, we schedule this job at the time-slot t on machine i in the initial problem. By doing that, the load at any time-slot t on machine i in the initial problem equals the load of the machine (i, t) in the L_ν-norm problem. Therefore, the constructed schedule for the initial problem has cost C^ν where C is the cost of the schedule in the L_ν-norm problem. □

By Lemma 3, solving the smart grid problem with unit processing time jobs and identical machines is essentially solving the L_ν-norm problem. Hence, in our algorithm (Algorithm 1), we invoke the Azar-Epstein algorithm [3] in order to get an approximation algorithm for the latter. Roughly speaking, the Azar-Epstein algorithm consists of solving a relaxed convex program and rounding fractional solutions to integral ones using the standard scheme of Lenstra et al. [12]. Given a solution for the L_ν-norm problem, we reconstruct a feasible solution for the smart grid problem with approximation ratio of 2^ν.

Theorem 2. *Algorithm 1 achieves an approximation ratio of* 2^ν.

Algorithm 1. Approximation algorithm for the smart grid scheduling problem with unit processing time jobs and identical machines

```
1: $\Theta = \cup_{j=1}^{n}\{r_j + a \mid a = -n, \ldots, n\} \cup_{j=1}^{n} \{d_j + a \mid a = -n, \ldots, n\}$
2: Let $\Pi = \emptyset$ be the set of machines and $\mathcal{J} = \emptyset$ be the set of jobs
3: for each $t \in \Theta$ and each machine $i$ do
4:     Create a machine $(i,t)$ and $\Pi \leftarrow \Pi \cup \{(i,t)\}$
5: end for
6: for each job $j$ do
7:     Create a new job $j'$ with $p_{(i,t),j'} = h_j$ if $t \in [r_j, d_j)$, otherwise we have $p_{(i,t),j'} = +\infty$
8:     $\mathcal{J} \leftarrow \mathcal{J} \cup \{j'\}$
9: end for
10: Apply the Azar-Epstein algorithm [3] on instance $(\Pi, \mathcal{J})$.
11: Build the schedule for the smart grid problem as in Lemma 3.
```

Proof. By Lemma 3, we know that given an assignment of jobs for the L_ν-norm problem on unrelated machines of cost C, we can construct a schedule for the smart grid problem with a cost of C^ν in polynomial time. Thus we have $(OPT_L)^\nu = OPT_{SG}$ where OPT_L is the optimal cost of the L_ν-norm problem and OPT_{SG} is the optimal cost of the smart grid problem.

Besides, Azar-Epstein algorithm [3] is 2-approximation for the L_ν-norm problem. Therefore, we have $OPT_L \leq C \leq 2OPT_L$. Finally, by raising each term of the inequality by a power of ν, we have $(OPT_L)^\nu \leq C^\nu \leq 2^\nu (OPT_L)^\nu$, so $OPT_{SG} \leq C^\nu \leq 2^\nu OPT_{SG}$. The theorem follows. $\square$

4 Concluding Remarks

In the paper, we have considered a general model of demand-response management in Smart Grid. We have given a competitive algorithm which is optimal (up to a constant factor) in typical settings. Our algorithm is robust to arbitrary demands and so enables the flexibility on the choices of clients in shaping their demands. The paper gives rise to several directions for future investigations. First, in the scheduling aspect, it would be interesting to consider problems in the general model with additional requirements such as precedence constraints, etc. Secondly, in the game theory aspect, designing pricing schemes that allow clients to react rationally while maintaining the efficiency in the energy consumption has received particular interests from both theoretical and practical studies in Smart Grid. Through the primal-dual view point, dual variables can be interpreted as the payments of clients. An interesting direction is to design a pricing scheme based on primal-dual approaches.

Acknowledgement. We thank Prudence W. H. Wong for insightful discussions and anonymous reviewers for useful comments that helps to improve the presentation.

References

1. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
2. Alford, R., Dean, M., Hoontrakul, P., Smith, P.: Power systems of the future: the case for energy storage, distributed generation, and microgrids. Zpryme Research & Consulting, Technical report (2012)
3. Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. In: Proceedings 37th Annual ACM Symposium on Theory of Computing, pp. 331–337 (2005)
4. Bell, P.C., Wong, P.W.H.: Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. J. Comb. Optim. **29**(4), 739–749 (2015)
5. Burcea, M., Hon, W., Liu, H.H., Wong, P.W.H., Yau, D.K.Y.: Scheduling for electricity cost in a smart grid. J. Sched. **19**(6), 687–699 (2016)
6. Chen, C., Nagananda, K., Xiong, G., Kishore, S., Snyder, L.V.: A communication-based appliance scheduling scheme for consumer-premise energy management systems. IEEE Trans. Smart Grid **4**(1), 56–65 (2013)
7. Cohen, J., Dürr, C., Thang, N.K.: Smooth inequalities and equilibrium inefficiency in scheduling games. In: Goldberg, P.W. (ed.) WINE 2012. LNCS, vol. 7695, pp. 350–363. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35311-6_26
8. Fang, K., Uhan, N.A., Zhao, F., Sutherland, J.W.: Scheduling on a single machine under time-of-use electricity tariffs. Ann. OR **238**(1–2), 199–227 (2016)
9. Feng, X., Xu, Y., Zheng, F.: Online scheduling for electricity cost in smart grid. In: Lu, Z., Kim, D., Wu, W., Li, W., Du, D.-Z. (eds.) COCOA 2015. LNCS, vol. 9486, pp. 783–793. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26626-8_58
10. Hamilton, K., Gulhar, N.: Taking demand response to the next level. IEEE Power Energy Mag. **8**(3), 60–65 (2010)
11. Koutsopoulos, I., Tassiulas, L.: Control and optimization meet the smart power grid: scheduling of power demands for optimal energy management. In: e-Energy, pp. 41–50. ACM (2011)
12. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**(1), 259–271 (1990)
13. Liu, F., Liu, H.H., Wong, P.W.H.: Optimal nonpreemptive scheduling in a smart grid model. In: Proceedings 27th Symposium on Algorithms and Computation, vol. 64, pp. 53:1–53:13 (2016)
14. Liu, F., Liu, H.H., Wong, P.W.H.: Optimal nonpreemptive scheduling in a smart grid model. CoRR abs/1602.06659 (2016). http://arxiv.org/abs/1602.06659
15. Lui, T.J., Stirling, W., Marcy, H.O.: Get smart. IEEE Power Energy Mag. **8**(3), 66–78 (2010)
16. Maharjan, S., Zhu, Q., Zhang, Y., Gjessing, S., Basar, T.: Dependable demand response management in the smart grid: a stackelberg game approach. IEEE Trans. Smart Grid **4**(1), 120–132 (2013)
17. Roughgarden, T.: Intrinsic robustness of the price of anarchy. J. ACM **62**(5), 32 (2015)
18. Salinas, S., Li, M., Li, P.: Multi-objective optimal energy consumption scheduling in smart grids. IEEE Trans. Smart Grid **4**(1), 341–348 (2013)

19. Thang, N.K.: Online primal-dual algorithms with configuration linear programs. CoRR abs/1708.04903 (2017)
20. US Department of Energy: The smart grid: an introduction (2009). https://energy.gov/oe/downloads/smart-grid-introduction-0
21. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382. IEEE Computer Society (1995)

An Average-Case Lower Bound Against ACC^0

Ruiwen Chen, Igor C. Oliveira(✉), and Rahul Santhanam

Department of Computer Science, University of Oxford, Oxford, UK
{ruiwen.chen,igor.carboni.oliveira,rahul.santhanam}@cs.ox.ac.uk

Abstract. In a seminal work, Williams [22] showed that NEXP (non-deterministic exponential time) does not have polynomial-size ACC^0 circuits. Williams' technique inherently gives a worst-case lower bound, and until now, no average-case version of his result was known. We show that there is a language L in NEXP and a function $\varepsilon(n) = 1/\log(n)^{\omega(1)}$ such that no sequence of polynomial size ACC^0 circuits solves L on more than a $1/2+\varepsilon(n)$ fraction of inputs of length n for all large enough n. Complementing this result, we give a nontrivial pseudo-random generator against polynomial-size $\mathsf{AC}^0[6]$ circuits. We also show that learning algorithms for quasi-polynomial size ACC^0 circuits running in time $2^n/n^{\omega}(1)$ imply lower bounds for the randomised exponential time classes RE (randomized time $2^{O(n)}$ with one-sided error) and $\mathsf{ZPE}/1$ (zero-error randomized time $2^{O(n)}$ with 1 bit of advice) against polynomial size ACC^0 circuits. This strengthens results of Oliveira and Santhanam [15].

Keywords: Circuit lower bounds · Average-case complexity
Pseudorandomness · Learning and natural properties

1 Motivation and Background

Significant advances in unconditional lower bounds are few and far between, specially in non-monotone boolean circuit models. In the 80s, there was substantial progress in proving circuit lower bounds for AC^0 (constant-depth circuits with unbounded fan-in AND and OR gates) [2,8,11,24] and $\mathsf{AC}^0[p]$ (AC^0 circuits extended with MOD_p gates) for p prime [16,19]. But even the case of $\mathsf{AC}^0[m]$ with m composite has remained little understood after decades of investigation, despite our expectation that MOD_m gates do not have much computational power.

In a seminal paper from a few years ago, Williams [22] proved a super-polynomial lower bound against ACC^0 (constant-depth circuits with unbounded fan-in AND, OR and MOD_m gates, for a fixed but arbitrary m) using a new lower bound technique: *the algorithmic method*. This result represents exciting progress on circuit lower bounds after a long gap. However, it has a couple of drawbacks when compared to previous lower bounds.

First, while previous lower bounds were for *explicit* functions, i.e., functions in P (deterministic polynomial time), Williams' lower bound is only known to hold for functions in NEXP [22], or in closely related classes [23]. (We note that even proving a lower bound for these much larger classes had been a longstanding open problem.) Unfortunately, the algorithmic method of Williams does not seem to be easily adaptable to give lower bounds for explicit functions.

Second, previous lower bounds and their subsequent extensions worked also in the *average case* setting, i.e., they showed that there were explicit functions which cannot be computed by polynomial-size circuits on significantly more than half the inputs of any input length. In other words, even computing the function correctly on a random input is hard. Williams' lower bound, on the other hand, only seems to give a worst-case lower bound, meaning that any polynomial-size family of ACC^0 circuits is only guaranteed to fail to compute the hard function in NEXP on a negligible fraction of inputs of length n, for infinitely many n. The question of strengthening the existing worst-case ACC^0 lower bound to the average case has been recently posed and discussed in [1].

2 Results and Techniques

Our main result addresses this second drawback of Williams' lower bound, and strengthen the main result from [22] to an average-case lower bound.

Theorem 1 (An average-case lower bound against ACC^0). *There is a function $\varepsilon(n) = 1/\log(n)^{\omega(1)}$ such that the following holds. There is a language L in NEXP such that for any polynomial-size family $\{C_n\}$ of ACC^0 circuits, there are infinitely many n such that C_n computes L correctly on at most a $1/2+\varepsilon(n)$ fraction of inputs of length n.*

Our proof of Theorem 1 in fact gives a much smaller upper bound on $\varepsilon(n)$, but stating this bound is a bit technical, so we defer it to Sect. 3.

Before sketching the main ideas behind the proof of Theorem 1, we attempt to explain why the original proof based on the algorithmic method fails to give an average-case lower bound. Williams' proof [22] employs an indirect diagonalization technique. The technique exploits Williams' algorithm solving satisfiability of poly-size ACC^0 circuits in time $2^{n-\omega(\log(n))}$. Assume, for the sake of contradiction, that $\mathsf{NTIME}(2^n)$ does have ACC^0 circuits of polynomial-size. It can be shown from this assumption that languages in $\mathsf{NTIME}(2^n)$ have *succinct witnesses*, i.e., YES instances have witnesses which can be represented succinctly by ACC^0 circuits of size $\mathsf{poly}(n)$. Now we can use the following guess-and-check procedure to compute any $L \in \mathsf{NTIME}(2^n)$ in non-deterministic time $2^n/n^{\omega(1)}$, contradicting the non-deterministic time hierarchy theorem. We guess a poly-size ACC^0 circuit encoding a witness for the instance, and then check that this circuit indeed encodes a witness by using the satisfiability algorithm for ACC^0 circuits. From the fact that the satisfiability algorithm runs in time $2^n/n^{\omega(1)}$, it follows that this guess-and-check procedure runs in time $2^n/n^{\omega(1)}$, giving the desired contradiction to the non-deterministic time hierarchy theorem. (This is just a high-level description – we refer to [22] for more details.)

A crucial step in the above proof is to use the assumption that NEXP is in poly-size ACC^0 to get succinct witnesses for NEXP languages. This step simply does not work if our assumption is that NEXP is in poly-size ACC^0 on average rather than in the worst case, and the proof fails completely.

It seems difficult to adapt the algorithmic method to get an average-case lower bound, so we try a different approach. A popular paradigm in complexity-theoretic pseudorandomness is *hardness amplification*: transforming a function that is worst-case hard for some class of circuits to a function that is average-case hard for the same class of circuits. Williams' result gives us a worst-case lower bound against polynomial-size ACC^0 circuits. Can we use hardness amplification to derive an average-case lower bound from this?

There are a couple of obstacles we need to overcome to make this approach work. First, hardness amplification typically requires that the class of circuits against which we are performing amplification can compute the Majority function [18]. We are trying to show an average-case lower bound against ACC^0 circuits, and we do not believe that poly-size ACC^0 circuits can compute Majority. However, while this does preclude us from amplifying to hardness $1/2-1/\mathsf{poly}(n)$ (i.e., showing that any circuits computing the function must fail on a $1/2-1/\mathsf{poly}(n)$ fraction of inputs) in a black-box way, we can still hope for weaker hardness amplification results which get us hardness $1/2-o(1)$. Indeed, using the connection between hardness amplification and list-decodable error-correcting codes due to Sudan et al. [21], hardness amplification procedures are known [9,10] which are applicable in our setting.

Second, and more problematically, the hard function we begin with is in NEXP, and we would like our new function resulting from hardness amplification also to be in NEXP. If we were to do hardness amplification in a black-box way starting from a NEXP function, the amplification needs to be *monotone*, and it is not hard to see that black-box monotone hardness amplification cannot even amplify worst-case hardness to hardness 0.99.[1]

To overcome this obstacle, we use instead a later result of Williams [23], where he shows that his lower bound [22] also holds for a function in $(\mathsf{NE}\cap\mathsf{coNE})/1$, i.e., both in $\mathsf{NE}=\mathsf{NTIME}[2^{O(n)}]$ and $\mathsf{coNE}=\mathsf{coNTIME}[2^{O(n)}]$, but with 1 bit of advice depending only on the input length. The advantage of starting from this later result of Williams is that when standard hardness amplification is applied, the resulting function *stays* in $(\mathsf{NE}\cap\mathsf{coNE})/1$.

This still leaves the problem of eliminating the 1 bit of advice in the upper bound. Doing this in a naive way would stop us from achieving hardness less than 3/4, but we show how to eliminate the advice with a negligible loss in our hardness parameter. This concludes our high-level description of the proof of Theorem 1.

[1] This corresponds to monotone error-correcting codes, which cannot have good distance. We refer to [4] for more details.

A natural question that arises when we have an average-case lower bound against a circuit class is whether we can construct *pseudo-random generators* (PRG) against the circuit class. An influential paradigm of [13], motivated by a similar paradigm in the cryptographic setting, shows how to transform average-case hardness into pseudorandomness, and conversely. However, this is only applicable when we have a hardness parameter $\varepsilon(n) \leqslant 1/n^{\Omega(1)}$, which Theorem 1 fails to achieve.

More recent work of [7] studies how to derive pseudorandomness from average-case hardness in cases where the hardness is not strong enough to apply [13]. It is shown in [7] that when the hard function has a property known as *resamplability* (a certain form of random self-reducibility), it is possible to get low-stretch pseudorandom generators with error $o(1)$ even under the weaker assumption that $\varepsilon(n) = o(1)$. We cannot directly apply their result in our setting because it is unclear if our hard function in NEXP satisfies the resamplability property.

However, by a win-win analysis and a result from [7], we are able to get a low-stretch pseudo-random generator against $\mathsf{AC}^0[6]$.[2] Ideally, we would like this generator to be computable in deterministic exponential time, but because our hard function for ACC^0 is in $(\mathsf{NE}\cap\mathsf{coNE})/1$, we are only able to get computability in *strong non-deterministic linear exponential time with 1 bit of advice.*[3]

Theorem 2 (A pseudo-random generator against $\mathsf{AC}^0[6]$). *For every depth $d \geqslant 1$ and $\delta > 0$, there is a sequence of functions $\{G_n\}$ computable in $(\mathsf{NE}\cap\mathsf{coNE})/1$, where each $G_n : \{0,1\}^{\ell} \to \{0,1\}^n$ has seed length $\ell(n) = n - n^{1-\delta}$, for which the following holds. Let $\{C_n\}$ be a sequence of $\mathsf{AC}^0[6]$ circuits, where each C_n has depth $\leqslant d$ and size $\leqslant n^d$. Then, for infinitely many values of n,*

$$\left| \mathbf{Pr}_{y\in\{0,1\}^{\ell}}[C_n(G_n(y)) = 1] - \mathbf{Pr}_{x\in\{0,1\}^{n}}[C_n(x) = 1] \right| \leqslant o(1).$$

We observe that, using the pseudo-random generator in Theorem 2, we can get an alternative proof of a variant of Theorem 1. Since this is obtained in a somewhat more indirect way, we do not discuss it further.

There are a couple of directions in which we could aspire to strengthen these results. First, in Theorem 1, we might hope to get a hardness parameter $\varepsilon(n) = 1/n^{\Omega(1)}$, or even $\varepsilon(n) = 1/n^{\omega(1)}$. Indeed, we are able to obtain an analogous result with $\varepsilon(n) = 1/n^{\Omega(1)}$, but for a hard function in E^{NP} instead of NEXP (see Theorem 9 in Sect. 3.4). Nevertheless, getting even stronger results seems to be a difficult task using existing techniques, for the following reason. Even for the substantially simpler case of $\mathsf{AC}^0[p]$ circuits, when p is prime, we do not know

[2] We stick to modulo 6 gates mostly for simplicity. Theorem 2 can be extended to any modulus m for which the results from [7] hold.

[3] In other words, the non-deterministic algorithm, when given the correct advice bit (that only depends on the input length parameter), outputs either "abort" of the correct string, and outputs the correct string in at least one computation path. We refer to Sect. 3.1 for more details.

how to get $\varepsilon(n) = o(1/\sqrt{n})$ for an explicit function, and showing a stronger hardness result is a long-standing open problem (cf. [20]).

Second, we could hope to get a PRG computable in *deterministic* linear exponential time in Theorem 2. But this would imply that EXP is hard on average for poly-size $\mathsf{AC}^0[6]$ circuits, and so far we have been unable to show even *worst-case* hardness against poly-size $\mathsf{AC}^0[6]$ for EXP. This brings us back to the first drawback in Williams' algorithm technique, discussed in Sect. 1, and which we further explore now.

While substantially improving the explicitness in Williams' lower bounds [22,23] and in Theorem 1 remains a major challenge, [14] recently introduced another approach that could lead to further progress in this direction. They considered a *learning-theoretic* analogue of Williams' approach. While Williams derives circuit lower bounds from circuit satisfiability algorithms that are "non-trivial" in that they beat the naive brute force search algorithm, [14] show implications for circuit lower bounds from learning algorithms that are similarly non-trivial in that they beat a brute force search approach.

We say that a randomized learning algorithm is a *non-trivial* learner for a circuit class $\mathcal{C}$ if it runs in time bounded by $2^n/n^{\omega(1)}$. For concreteness and simplicity, we consider learning algorithms that make membership queries, and that learn under the uniform distribution with error at most $1/n$ and with failure probability at most $1/n$.

For convenience, we use $\mathsf{ACC}^0_{d,m}(s(n))$ to denote the class of boolean functions computable by depth-d ACC^0 circuits over a fixed modulo m and of size $\leqslant s(n)$. The following connection between learning algorithms and non-uniform lower bounds was established in [14].

Proposition 1 **(REXP lower bounds from learning sub-exponential size ACC^0 circuits [14]).** *If for every depth $d \geqslant 1$ and modulo $m \geqslant 1$ there is $\varepsilon > 0$ such that $\mathsf{ACC}^0_{d,m}(2^{n^\varepsilon})$ can be learned in non-trivial time, then $\mathsf{REXP} \nsubseteq \mathsf{ACC}^0$.*

Recall that $\mathsf{REXP} \subseteq \mathsf{NEXP}$ is the class of languages decided by one-sided randomized exponential time algorithms, and that under standard derandomization hypotheses, $\mathsf{REXP} = \mathsf{EXP}$.[4] Consequently, Proposition 1 offers a potential path to more explicit (worst-case) ACC^0 lower bounds via the design of non-trivial learning algorithms, and it can be seen as another instantiation of the algorithmic method.[5]

However, note that the learnability of *sub-exponential* size circuits is a strong assumption. Indeed, by the Speedup Lemma of [14], it implies that polynomial size ACC^0 circuits can be learned in *quasi-polynomial* time, a result that is only known to hold for AC^0 and $\mathsf{AC}^0[p]$ circuits [5]. Ideally, we would like to get stronger and more explicit lower bounds from much weaker assumptions.

[4] For a concrete example of the benefits of improving an NEXP lower bound to randomized exponential time classes such as REXP, we refer the reader to [15].

[5] The design of concrete non-trivial learning algorithms for some circuit classes and in some alternative but related learning models has been recently investigated in [17].

The proof of Proposition 1 relies on a variety of techniques from complexity theory. An important element in the argument is the use of Williams' unconditional proof that $\mathsf{NEXP} \nsubseteq \mathsf{ACC}^0$. This lower bound is employed as a *black-box* in the argument from [14], and in Sect. 8 of the same work, the authors speculate about the possibility of establishing stronger connections between non-trivial algorithms and lower bounds by combining ideas from different frameworks.

We present a new application of the interaction between the learning framework of [14], and the satisfiability framework of Williams [22,23]. We combine *the proofs* of existing connections between non-trivial algorithms and non-uniform lower bounds, and establish the following result.

Theorem 3 (Stronger connection between ACC^0-learnability and lower bounds). *Assume that for every fixed choice of parameters $d, m, c \geqslant 1$, the class $\mathsf{ACC}^0_{d,m}(n^{(\log n)^c})$ can be non-trivially learned. Then,*

$$\mathsf{RTIME}[2^{O(n)}] \nsubseteq \mathsf{ACC}^0(n^{\log n}) \quad \textit{and} \quad \mathsf{ZPTIME}[2^{O(n)}]/1 \nsubseteq \mathsf{ACC}^0(n^{\log n}).$$

We note that the worst-case lower bound for $\mathsf{ZPTIME}[2^{O(n)}]/1$ in Theorem 3 can be strengthened to an average-case lower bound using the same technique as in the proof of Theorem 1.

Observe that this result strengthens Proposition 1 in a few ways. The assumption is considerably weaker, and the lower bound is quantitatively stronger. In addition, it provides a lower bound for *zero-error* randomized computations with one bit of advice, while in Proposition 1 the randomized algorithm computing the hard function makes mistakes. Interestingly, Theorem 3 is not known to hold for larger circuit classes, and its proof explores specific results about ACC^0 circuits in a crucial way.

We note that there is a connection between non-trivial algorithms and non-uniform lower bounds for ZPEXP, but it assumes the existence of P-natural properties useful against *sub-exponential* size circuits (see Theorem 44 from [14], and also [12]). Although in Theorem 3 the uniformity over the hard language is not as strong (i.e., REXP and $\mathsf{ZPEXP}/1$ versus ZPEXP), it almost matches the uniformity condition, while its assumption is considerably weaker.

We sketch in the next section the proof of Theorem 1. Due to space limitations, we refer to the full version of the paper [6] for more details about our results.

3 Proof of Theorem 1

3.1 Notation for Complexity Classes and Circuit Classes

Let $\mathsf{TIME}[t(n)]$ be the classes of languages decided by deterministic Turing machines (TM) running in time $O(t(n))$, and let $\mathsf{NTIME}[t(n)]$ be the class of languages decided by non-deterministic Turing machines (NTM) running in time $O(t(n))$. We use standard notions of complexity classes, such as P, NP, EXP, NEXP, etc. In particular, $\mathsf{E} = \mathsf{TIME}[2^{O(n)}]$, $\mathsf{NE} = \mathsf{NTIME}[2^{O(n)}]$, and

L is the class of languages computable in (uniform) logarithmic space. A function $t : \mathbb{N} \to \mathbb{N}$ is time-constructible if there is a TM M, which on input 1^n outputs $t(n)$ in time $O(t(n))$. We sometimes informally use the term algorithm instead of Turing machines. We refer to a textbook such as [3] for more background in complexity theory.

A *strong non-deterministic Turing machine* (SNTM) is a NTM where each branch of the computation has one of three possible outputs: 0, 1, and '?'. We say that a SNTM M decides a language L if the following *promise* holds: if $x \in L$, each branch ends with 1 or '?', and at least one branch ends with 1; if $x \notin L$, each branch ends with 0 or '?', and at least one branch ends with 0. It is easy to see that a language $L \in \mathsf{NE} \cap \mathsf{coNE}$ if and only if L is decided by a SNTM in time $2^{O(n)}$. When we say that a sequence of functions $G_n \colon \{0,1\}^\ell \to \{0,1\}^n$ is computed by a SNTM M, we formally mean that the language $L_G \subseteq \{0,1\}^\star$ that encodes $\{G_n\}$ is computed by M, where L_G is defined in a natural way. For concreteness, we let L_G be the set of strings encoding tuples $\langle 1^n, y, i, b\rangle$, where $b \in \{0,1\}$, $y \in \{0,1\}^{\ell(n)}$, $i \in [n]$, and $G_n(y)_i = b$. We assume that the tuples obtained from each choice of the parameter n have all the same length as strings in $\{0,1\}^\star$ (this is relevant when defining computation with advice below).

We define *advice classes* as follows. For a deterministic or non-deterministic uniform complexity class $\mathcal{C}$ and a function $\alpha(n)$, the class $\mathcal{C}/\alpha(n)$ is the set of languages L such that there is a language $L' \in \mathcal{C}$ and a sequence of strings $\{a_n\}$ with $|a_n| = \alpha(n)$ which satisfy that $L(x) = L'(x, a_{|x|})$ for all strings $x \in \{0,1\}^\star$.

For *semantic classes* $\mathcal{C}$ (such as BPP, $\mathsf{NE} \cap \mathsf{coNE}$, etc.) with advice, we only require the *promise condition* for the class $\mathcal{C}$ to hold when the correct advice is given. For example, a language L is in $(\mathsf{NE} \cap \mathsf{coNE})/\alpha(n)$ if there is a SNTM M running in time $2^{O(n)}$ and a sequence of advice strings $\{a_n\}$ with $|a_n| = \alpha(n)$ such that, on each input x, the computation paths of $M(x, a_{|x|})$ satisfy the promise condition in the definition of SNTMs. Note that M running with incorrect advice may not satisfy the promise on its branches.

We also define *infinitely often classes.* For a (syntactic) deterministic or non-deterministic class $\mathcal{C}$, the class $\mathsf{i.o.}\mathcal{C}$ is the set of languages L for which there is a language $L' \in \mathcal{C}$ such that, for infinitely many values of n, $L \cap \{0,1\}^n = L' \cap \{0,1\}^n$. For a semantic class $\mathcal{C}$, we relax the definition, and let $\mathsf{i.o.}\mathcal{C}$ be the class of languages L decided by a Turing machine M such that, for infinitely many input lengths n, M is of type $\mathcal{C}$ on inputs of length n (i.e., it satisfies the corresponding promise). Note that M might not be of type $\mathcal{C}$ on other input lengths.

We use standard notation for circuit classes. In particular, AC^0 is the class of circuit families of constant depth and polynomial size, with AND, OR, and NOT gates, where AND and OR gates have unbounded fan-in. $\mathsf{AC}^0[m]$ extends AC^0 by allowing unbounded fan-in MOD_m gates, where m is fixed, and $\mathsf{ACC}^0 \overset{\text{def}}{=} \bigcup_m \mathsf{AC}^0[m]$ (we often write $\mathsf{AC}^0[m]$ and $\mathsf{ACC}^0[m]$ interchangeably). For convenience, we use $\mathcal{C}_d(s)$ to restrict a circuit class to circuits of depth $\leqslant d$ and size $\leqslant s$. We often deliberately conflate a class of circuit families with the class of languages computed by the circuit families. These circuit families are all *non-uniform*, unless otherwise stated.

We say that a language L is $\gamma(n)$-*hard* for a circuit class $\mathcal{C}$ if for each $L' \in \mathcal{C}$ and for infinitely many values of n, $\mathbf{Pr}_{x \in \{0,1\}^n}[L(x) = L'(x)] \leqslant 1 - \gamma(n)$. Finally, a class Γ is $\gamma(n)$-hard for $\mathcal{C}$ if there is a language in Γ that is $\gamma(n)$-hard for $\mathcal{C}$.

3.2 Background on ACC^0 Lower Bounds

We recall the following ACC^0 circuit lower bounds.

Theorem 4 **([22]).** *For every $d \geqslant 1$ and $m \geqslant 1$, there is a $\delta > 0$ and a language in E^{NP} that is not computable by a sequence of $\mathsf{ACC}^0[m]$ circuits of depth d and size $O(2^{n^\delta})$.*

Theorem 5 **([23]).** *There is a language in $(\mathsf{NE} \cap \mathsf{coNE})/1$ that does not admit ACC^0 circuits of size $O(n^{\log n})$.*

In order to prove Theorem 1 and its extensions, we need a strengthening of Theorem 5. We use the following technical definitions. A function $f \colon \mathbb{N} \to \mathbb{N}$ is *sub-half-exponential* if for every fixed $k \geqslant 1$, $f(f(n^k)^k) \leqslant 2^{n^{o(1)}}$. Similarly, a function $g \colon \mathbb{N} \to \mathbb{N}$ is *sub-third-exponential* if for every fixed $k \geqslant 1$, $g(g(g(n^k)^k)^k) \leqslant 2^{n^{o(1)}}$. For instance, for a fixed integer $a \geqslant 1$, $g(n) \overset{\text{def}}{=} 2^{(\log n)^a}$ is sub-third-exponential.

By a more careful application of William's techniques, the following result can be established. We refer to the full version of the paper [6] for details.

Theorem 6 **(Sub-third-exponential lower bounds against ACC^0).** *$(\mathsf{NE} \cap \mathsf{coNE})/1$ does not have ACC^0 circuits of sub-third-exponential size.*

3.3 Tools: Error Correcting Codes and Hardness Amplification

We follow part of the terminology from [10]. The proof of Theorem 1 requires certain correcting codes that admit a uniform encoding procedure, but whose decoding can be non-uniform.

Definition 1 **(Local-list-decoding in error correcting codes).** *A family $\{C_M\}_M$ of functions $C_M \colon \{0,1\}^M \to \{0,1\}^N$ is a (d, L)*-locally-list-decodable code *if there is an oracle Turing machine D that takes an index $i \in [M]$, advice $a \in [L]$, and a random string r, and for which the following holds. For every input $x \in \{0,1\}^M$ and $y \in \{0,1\}^N$ for which $\Delta(C_M(x), y) \leqslant d$, there exists $a \in [L]$ such that, for all $i \in [M]$,*

$$\mathbf{Pr}_r[D^y(i, a, r) = x_i] > 9/10.$$

Here $\Delta(w_1, w_2) \in [0,1]$ is the relative hamming distance *between strings w_1 and w_2, and one should think of $N = N(M)$, $d = d(M)$, etc. as a sequence of parameters indexed by M. We say that a code of this form is* explicit *if it can be computed in time $\mathsf{poly}(N(M))$.*

We will need results on hardness amplification and constructions of efficient error correcting codes.

Definition 2 (Black-box hardness amplification). *A* $(1/2-\epsilon, \delta)$-black-box hardness amplification *from input length* k *to input length* n *is a pair* $(\mathsf{Amp}, \mathsf{Dec})$ *where* Amp *is an oracle Turing machine that computes a (sequence of) boolean function on* n *bits,* Dec *is a randomized oracle Turing machine on* k *bits which also takes an advice string of length* a*, and for which the following holds. For every pair of functions* $f\colon \{0,1\}^k \to \{0,1\}$ *and* $h\colon \{0,1\}^n \to \{0,1\}$ *such that*

$$\mathbf{Pr}_{x\sim\{0,1\}^n}[h(x) = \mathsf{Amp}^f(x)] > 1/2 + \epsilon,$$

there is an advice string $\alpha \in \{0,1\}^a$ *such that*

$$\mathbf{Pr}_{x\sim\{0,1\}^k,\mathsf{Dec}}[\mathsf{Dec}^h(x,\alpha) = f(x)] > 1 - \delta.$$

(We will also view Dec^h *as a non-uniform oracle boolean circuit. Observe that if* $\delta = 2^{-k}$ *then there is a way to fix the randomness and the advice string of* Dec^h *so that it correctly computes* f *on every input* $x \in \{0,1\}^k$*.*[6]*)*

The following is a well-known connection [21] between fully black-box hardness amplification and binary locally-list-decodable codes.

Theorem 7 (Connection between hardness amplification and local-list-decodable codes). *If there is a* $(1/2-\epsilon, L)$*-locally list decodable error-correcting code* $C\colon \{0,1\}^K \to \{0,1\}^N$ *with a corresponding decoder* D *then there is a* $(1/2-\epsilon, 2^{-k})$*-black-box hardness amplification procedure from length* $k = \log K$ *to length* $n = \log N$*, where* Amp *is defined by the encoder of* C*, and* Dec *is defined by the decoder* D *with advice length* $a = \log L$*.*

We need the following construction of list-decodable codes (and corresponding hardness amplification procedure).

Theorem 8 (Efficient construction of locally-list-decodable codes [9,10]). *For every* $\exp(-\Theta(\sqrt{\log M})) \leqslant \epsilon < 1/2$*, there is an explicit* $(1/2-\epsilon, \mathsf{poly}(1/\epsilon))$*-locally-list-decodable code* $C_M\colon \{0,1\}^M \to \{0,1\}^{\mathsf{poly}(M)}$ *with a local decoder that can be implemented by a family of constant-depth circuits of size* $\mathsf{poly}(\log M, 1/\epsilon)$ *using majority gates of fan-in* $\Theta(1/\epsilon)$ *and* AND/OR *gates of unbounded fan-in.*

Observe that it is possible to get an AC^0 decoder by a standard simulation of majority gates via large AC^0 circuits.

[6] Note that the process of amplifying the success probability of randomized algorithms and fixing the randomness can be done with only an AC^0 overhead on the overall complexity, since approximate majority functions can be computed in this circuit class.

Corollary 1 (Limited hardness amplification via constant-depth circuits of bounded size). *For every parameter* $\exp(-\Theta(\sqrt{\log M})) \leqslant \epsilon < 1/2$ *and each large enough constant d, there is an explicit* $(1/2-\epsilon, \mathsf{poly}(1/\epsilon))$*-locally-list-decodable code* $C_M\colon \{0,1\}^M \to \{0,1\}^{\mathsf{poly}(M)}$ *with a local decoder that can be implemented by* AC^0 *circuits of size* $\mathsf{poly}(\log M, \exp((1/\epsilon)^{O(1/d)}))$ *and depth at most d.*

Corollary 1 and the connection to hardness amplification are crucial results needed in the proof of Theorem 1 and its extensions. We will implicitly use these locally-list-decodable codes in order to amplify from worst-case hardness to average-case hardness.

3.4 The Proof of Theorem 1 and Its Extensions

We start off by showing a $(1/2 - 1/n^{\Omega(1)})$-hardness result for E^{NP}.

Theorem 9 (An average-case lower bound for E^{NP}). *For every* $d \geqslant 1$ *and* $m \geqslant 1$*, there is a* $\gamma > 0$ *and a language in* E^{NP} *that is* $(1/2 - 1/n^{\gamma})$*-hard for nonuniform* $\mathsf{AC}^0[m]$ *circuits of depth d and size* $2^{n^{\gamma}}$*.*

Proof. Given a sufficiently large $d \geqslant 1$ and a fixed modulo m, let $L^d \in \mathsf{E}^{\mathsf{NP}}$ be the language guaranteed to exist by Theorem 4, and $\delta = \delta(d,m) > 0$ be the corresponding constant. In other words, L^d is not computed by $\mathsf{AC}^0[m]$ circuits of depth d and size $2^{n^{\delta}}$ for infinitely many values of n. For a function $\epsilon' = \epsilon'(M') \geqslant \exp(-\Theta(\sqrt{\log M'}))$ to be fixed later in the proof, and d' sufficiently large (but smaller than d), let $\{C_{M'}\}$ be the sequence of explicit error-correcting codes provided by Corollary 1, where each $C_{M'}\colon \{0,1\}^{M'} \to \{0,1\}^N$, and $N(M') = M'^c$ for a fixed positive integer $c \geqslant 1$. Consider a new language $L^\star$ that depends on L^d and on $\{C_{M'}\}$, defined as follows. Given $x \in \{0,1\}^n$, if n is not of the form cm' for some $m' \in \mathbb{N}$, then x is not in $L^\star$. Otherwise, let $T \in \{0,1\}^{2^{m'}}$ be the truth-table of L^d on m'-bit inputs, and consider the codeword $C_{M'}(T) \in \{0,1\}^N$, where $M' = 2^{m'}$ and $N = M'^c = 2^{cm'} = 2^n$. Then $x \in L^\star$ if and only if the entry of $C_{M'}(T)$ indexed by x is 1. This completes the description of $L^\star$.

Given that $L^d \in \mathsf{E}^{\mathsf{NP}}$ and $C_{M'}$ can be computed in deterministic time $\mathsf{poly}(M')$, we can compute $L^\star$ in E^{NP} as follows. Let x be an input of length N, on which we wish to solve $L^\star$. First, check if $N = M'^c$ for some integer M'. If not, output 0. Otherwise, compute the truth table T of L^d on input length M' by running the E^{NP} machine for L^d on every possible input of length M'. Then compute $C_{M'}(T)$ and output the x'th bit of that string. The computation of T can be done in E^{NP} as it involves at most 2^N runs of an E^{NP} machine on inputs of length $\leqslant N$, and the computation of $C_{M'}(T)$ can be done in time $2^{O(N)}$ just using the efficiency guarantee for $C_{M'}$. Hence the procedure described above can be implemented in E^{NP}.

Now we show that $L^\star$ has the claimed average-case hardness. For $n = cm'$ and $M' = 2^{m'}$ as above, we set $\epsilon'(M') \overset{\text{def}}{=} 1/n^{2\gamma} \gg \exp(-\Theta(\sqrt{\log M'}))$, where $0 < \gamma < \delta/2$ is a sufficiently small constant. We claim that $L^\star$ cannot be computed with advantage larger than $1/n^\gamma$ on infinitely many input lengths by $AC^0[m]$ circuits of depth $\leqslant d$ and size $\leqslant 2^{n^\gamma}$. This follows by the properties of the code $C_{M'}$ and the connection to hardness amplification. Indeed, if for all large n of the form cm' the boolean function computed by $L^\star_n$ could be approximated by such circuits, by hardcoding their descriptions into the AC^0 local decoders provided by Corollary 1 it would follow that for all large n the language L^d is (worst-case) computable by $\mathsf{AC}^0[m]$ circuits of depth $\leqslant d$ and size $\leqslant 2^{n^\delta}$, a contradiction. (This last step crucially uses that γ is sufficiently small compared to the other parameters, and the size bound in Corollary 1.)

Next, we address the more difficult problem of showing an average-case lower bound for NEXP. We first establish a lower bound for $(\mathsf{NE} \cap \mathsf{coNE})/1$, and then show how to remove the advice.

Lemma 1. *$(\mathsf{NE} \cap \mathsf{coNE})/1$ is $(1/2 - 1/\log(t(n)))$-hard for ACC^0 circuits of size $t(n)$, for any (time-constructible) sub-third-exponential function $t(n)$.*

Proof. The argument follows the same high-level approach of Theorem 9, so we use the same notation and only describe the relevant differences. By Theorem 6, there is a language $L \in (\mathsf{NE} \cap \mathsf{coNE})/1$ that is not computable by ACC^0 circuits of sub-third-exponential size $t(n)$. Similarly, we define a language $L^\star$ obtained from L and the locally-list-decodable codes provided by Corollary 1. We need to make sure the new language is still computable in $(\mathsf{NE} \cap \mathsf{coNE})/1$, and explain the choice of parameters in the construction.

Since $L \in (\mathsf{NE} \cap \mathsf{coNE})/1$, there is a strong non-deterministic Turing machine (SNTM) S with one bit of advice computing L. Let the advice sequence for M be $\alpha(\cdot)$, where $|\alpha(n)| = 1$ for all $n \in \mathbb{N}$. We define an SNTM S' with one bit of advice computing $L^\star$. S' acts as follows on input x of length N. It checks if $N = M'^c$ for some integer M'. If not, it rejects. If yes, it simulates S with advice $\alpha(M')$ on each input of length M'. The advice $\alpha(M')$ is the advice bit for S' - note that N completely determines M' and hence $\alpha(M')$. If any of these simulations outputs '?', it outputs '?' and halts. If all of these simulations output non-'?' values, S' uses the results of its simulations of S to compute the truth table T of L on input length M', and applies the mapping $C_{M'}$ to this string. It then outputs the bit with index x of the resulting string.

We need to show that S' is an SNTM with one bit of advice deciding $L^\star$ correctly in time $2^{O(N)}$. By definition of S', and using the fact that S is an SNTM with one bit of advice, we have that whenever S' computes a string T, this is the correct truth table of L on inputs of length M', if S' uses advice $\beta(N) = \alpha(M')$. Moreover, this happens on at least one computation path of S, using the fact that S' is an SNTM with one bit of advice. On any such computation path, the correct value $L^\star(x)$ is output, as S' is completely deterministic after computing T, and using the definition of $L^\star$. The time taken by S' is $2^{O(n)}$, as it simulates S on inputs of length $\leqslant N$ at most 2^N times, and using the efficiency guarantee on $C_{M'}$.

Finally, we sketch the choice of parameters in the hardness amplification, which correspond to the parameters in the construction of $L^\star$ via the error-correcting code provided by Corollary 1. Following the notation in the proof of Theorem 9, we let $\epsilon(M')$ be of order $1/\log(t(\beta n)^{\beta})$, where $\beta > 0$ is sufficiently small. Under this definition, observe that the circuit complexity overhead coming from the decoder in the analysis of the average-case hardness of $L^\star$ is at most $\mathsf{poly}(n, \exp(1/\epsilon')) \leqslant \mathsf{poly}(n, \exp(\log t(\beta n)^{\beta})) \leqslant t(n)^{\gamma}$, for a fixed but arbitrarily small $\gamma > 0$ that depends on β. This implies that $L^\star$ is $1/\log t(\Omega(n))^{\Omega(1)}$-hard against circuits of size $t(n)^{\Omega(1)}$. Since our original sub-third-exponential function $t(n)$ was arbitrary and after composition with polynomials a function remains in this class, the proof is complete.

We give a generic way to eliminate advice from the upper bound for average-case hardness results.

Lemma 2. *If* $\mathsf{NE}/1$ *is* $(1/2 - \epsilon(n))$*-hard for* $\mathcal{C}$ *circuits of size* $s(n)$*, then* NE *is* $(1/2 - \epsilon(\lfloor n/2 \rfloor))$*-hard for* $\mathcal{C}$ *circuits of size* $s(\lfloor n/2 \rfloor)$*.*

Proof. Let L be a language in $\mathsf{NE}/1$ which is $(1/2 - \epsilon)$-hard for $\mathcal{C}$ circuits of size $s(n)$. Suppose L is decided by a NTM M running in nondeterministic time $2^{O(n)}$ and taking advice bits $\{b_n\}$, where $|b_n| = 1$. In other words, for every string x, we have $L(x) = M(x, b_{|x|})$.

Define a new language L' as follows. We divide the input string z in the middle, and denote it by xy, where either $|y| = |x|$ (when $|z|$ is even) or $|y| = |x| + 1$ (when $|z|$ is odd). Then we decide by running M on the first half x, using an advice bit which depends only on the length of y. More precisely, we let

$$L'(xy) \overset{\text{def}}{=} \begin{cases} M(x, 0), & \text{if } |y| = |x|; \\ M(x, 1), & \text{if } |y| = |x| + 1. \end{cases}$$

Obviously, L' is in NE by simulating M.

We show that if L_n is hard to approximate, then either L'_{2n} or L'_{2n+1} is also hard to approximate. For contradiction, suppose that both L'_{2n} and L'_{2n+1} can be computed correctly on more than a $1/2 + \epsilon$ fraction of inputs by circuits of size s. If the advice bit $b_n = 0$, let C_0 be a circuit of size s such that $\mathbf{Pr}_{xy}[L'_{2n}(xy) = C_0(xy)] > 1/2 + \epsilon$, where x and y are both chosen independently and uniformly at random from $\{0,1\}^n$. By an averaging argument, there is a specific $y^\star$ such that by fixing $y = y^\star$, $\mathbf{Pr}_x[L'_{2n}(xy^\star) = C_0(xy^\star)] > 1/2 + \epsilon$. Note also that, since $b_n = 0$, we have that for all x of length n, $L'_{2n}(xy^\star) = M(x, 0) = L_n(x)$. Thus $\mathbf{Pr}_x[L_n(x) = C_0(xy^\star)] > 1/2 + \epsilon$. That is, we can use C_0 to approximate L_n by fixing the second half of the inputs to $y^\star$. In the other case where the advice bit $b_n = 1$, we can use the approximate circuit for L'_{2n+1} to approximate L_n in the same way. As a consequence, if L_n is $(1/2 - \epsilon(n))$-hard for $\mathcal{C}$ circuits of size s, then either L'_{2n} or L'_{2n+1} is also $(1/2 - \epsilon(n))$-hard for $\mathcal{C}$ circuits of size s.

Finally, since there are infinitely many input lengths n such that L_n is $(1/2 - \epsilon(n))$-hard for $\mathcal{C}$ circuits of size $s(n)$, there are also infinitely many input lengths n such that L'_n is $(1/2 - \epsilon(\lfloor n/2 \rfloor))$-hard for $\mathcal{C}$ circuits of size $s(\lfloor n/2 \rfloor)$. This completes the proof.

Finally, by combining the previous two lemmas, we get the following strengthened version of Theorem 1.

Theorem 10 (An average-case lower bound for NE against sub-third-exponential size ACC^0). NE *is* $(1/2 - 1/\log t(n))$*-hard for* ACC^0 *circuits of size* $t(n)$ *when* $t(n)$ *is time-constructible and sub-third-exponential.*

Acknowledgements. We would like to thank Marco Carmosino for posing the question of proving average-case hardness against ACC^0, and for useful discussions. This work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2014)/ERC Grant Agrement no. 615075.

References

1. TCS Stack Exchange: How powerful is ACC^0 circuit class in average case? https://cstheory.stackexchange.com/q/37232. Accessed 27 Sept 2017
2. Ajtai, M.: Σ_1^1-formulae on finite structures. Ann. Pure Appl. Logic **24**(1), 1–48 (1983). https://doi.org/10.1016/0168-0072(83)90038-6
3. Arora, S., Barak, B.: Complexity Theory: A Modern Approach. Cambridge University Press, Cambridge (2009)
4. Buresh-Oppenheim, J., Kabanets, V., Santhanam, R.: Uniform hardness amplification in NP via monotone codes. In: Electronic Colloquium on Computational Complexity (ECCC) TR06-154 (2006). https://eccc.weizmann.ac.il/eccc-reports/2006/TR06-154/
5. Carmosino, M.L., Impagliazzo, R., Kabanets, V., Kolokolova, A.: Learning algorithms from natural proofs. In: Conference on Computational Complexity (CCC), pp. 10:1–10:24 (2016). https://doi.org/10.4230/LIPIcs.CCC.2016.10
6. Chen, R., Oliveira, I.C., Santhanam, R.: An average-case lower bound against ACC^0. In: Electronic Colloquium on Computational Complexity (ECCC) TR17-173 (2017). https://eccc.weizmann.ac.il/report/2017/173/
7. Fefferman, B., Shaltiel, R., Umans, C., Viola, E.: On beating the hybrid argument. Theory Comput. **9**, 809–843 (2013). https://doi.org/10.4086/toc.2013.v009a026
8. Furst, M., Saxe, J.B., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. Math. Syst. Theory **17**(1), 13–27 (1984). https://doi.org/10.1007/BF01744431
9. Goldwasser, S., Gutfreund, D., Healy, A., Kaufman, T., Rothblum, G.N.: Verifying and decoding in constant depth. In: Symposium on Theory of Computing (STOC), pp. 440–449 (2007). https://doi.org/10.1145/1250790.1250855
10. Gutfreund, D., Rothblum, G.N.: The complexity of local list decoding. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX/RANDOM -2008. LNCS, vol. 5171, pp. 455–468. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85363-3_36
11. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: Symposium on Theory of Computing (STOC), pp. 6–20 (1986). https://doi.org/10.1145/12130.12132
12. Impagliazzo, R., Kabanets, V., Volkovich, I.: The power of natural properties as oracles. In: Electronic Colloquium on Computational Complexity (ECCC) TR17-023 (2017). https://eccc.weizmann.ac.il/report/2017/023/
13. Nisan, N., Wigderson, A.: Hardness vs randomness. J. Comput. Syst. Sci. **49**(2), 149–167 (1994). https://doi.org/10.1016/S0022-0000(05)80043-1

14. Oliveira, I.C., Santhanam, R.: Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In: Computational Complexity Conference (CCC), pp. 18:1–18:49 (2017). https://doi.org/10.4230/LIPIcs.CCC.2017.18
15. Oliveira, I.C., Santhanam, R.: Pseudodeterministic constructions in subexponential time. In: Symposium on Theory of Computing (STOC), pp. 665–677 (2017). https://doi.org/10.1145/3055399.3055500
16. Razborov, A.A.: Lower bounds on the size of bounded-depth networks over the complete basis with logical addition. Math. Notes Acad. Sci. USSR **41**(4), 333–338 (1987)
17. Servedio, R., Tan, L.Y.: What circuit classes can be learned with non-trivial savings? In: Innovations in Theoretical Computer Science Conference (ITCS), pp. 1–23 (2017)
18. Shaltiel, R., Viola, E.: Hardness amplification proofs require majority. SIAM J. Comput. **39**(7), 3122–3154 (2010). https://doi.org/10.1137/080735096
19. Smolensky, R.: Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In: Symposium on Theory of Computing (STOC), pp. 77–82 (1987). https://doi.org/10.1145/28395.28404
20. Srinivasan, S.: On improved degree lower bounds for polynomial approximation. In: Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 201–212 (2013). https://doi.org/10.4230/LIPIcs.FSTTCS.2013.201
21. Sudan, M., Trevisan, L., Vadhan, S.P.: Pseudorandom generators without the XOR lemma. J. Comput. Syst. Sci. **62**(2), 236–266 (2001). https://doi.org/10.1006/jcss.2000.1730
22. Williams, R.: Nonuniform ACC circuit lower bounds. J. ACM **61**(1), 2:1–2:32 (2014). https://doi.org/10.1145/2559903
23. Williams, R.: Natural proofs versus derandomization. SIAM J. Comput. **45**(2), 497–529 (2016). https://doi.org/10.1137/130938219
24. Yao, A.C.: Separating the polynomial-time hierarchy by oracles (preliminary version). In: Symposium on Foundations of Computer Science (FOCS), pp. 1–10 (1985). https://doi.org/10.1109/SFCS.1985.49

Compressed Indexing with Signature Grammars

Anders Roy Christiansen and Mikko Berggren Ettienne(✉)

The Technical University of Denmark, Kongens Lyngby, Denmark
miet@dtu.dk

Abstract. The *compressed indexing problem* is to preprocess a string S of length n into a compressed representation that supports pattern matching queries. That is, given a string P of length m report all occurrences of P in S.

We present a data structure that supports pattern matching queries in $O(m + \mathsf{occ}(\lg\lg n + \lg^{\epsilon} z))$ time using $O(z\lg(n/z))$ space where z is the size of the LZ77 parse of S and $\epsilon > 0$ is an arbitrarily small constant, when the alphabet is small or $z = O(n^{1-\delta})$ for any constant $\delta > 0$. We also present two data structures for the general case; one where the space is increased by $O(z\lg\lg z)$, and one where the query time changes from worst-case to expected. These results improve the previously best known solutions. Notably, this is the first data structure that decides if P occurs in S in $O(m)$ time using $O(z\lg(n/z))$ space.

Our results are mainly obtained by a novel combination of a randomized grammar construction algorithm with well known techniques relating pattern matching to 2D-range reporting.

1 Introduction

Given a string S and a pattern P, the core problem of pattern matching is to report all locations where P occurs in S. Pattern matching problems can be divided into two: the algorithmic problem where the text and the pattern are given at the same time, and the data structure problem where one is allowed to preprocess the text (pattern) before a query pattern (text) is given. Many problems within both these categories are well-studied in the history of stringology, and optimal solutions to many variants have been found.

In the last decades, researchers have shown an increasing interest in the compressed version of this problem, where the space used by the index is related to the size of some compressed representation of S instead of the length of S. This could be measures such as the size of the LZ77-parse of S, the smallest grammar representing S, the number of runs in the BWT of S, etc. see e.g. [3,8–10,13,16,17]. This problem is highly relevant as the amount of highly-repetitive data increases rapidly, and thus it is possible to handle greater amounts of data by compressing it. The increase in such data is due to things like DNA sequencing, version control repositories, etc.

In this paper we consider what we call the *compressed indexing problem*, which is to preprocess a string S of length n into a compressed representation that supports fast *pattern matching queries*. That is, given a string P of length m, report all occ occurrences of substrings in S that match P.

Table 1 gives an overview of the results on this problem.

Table 1. Selection of previous results and our new results on compressed indexing. The variables are the text size n, the LZ77-parse size z, the pattern length m, occ is the number of occurrences and σ is the size of the alphabet. (The time complexity marked by † is expected whereas all others are worst-case)

Index	Space	Locate time	σ
Gagie et al. [9]	$O(z \lg(n/z))$	$O(m \lg m + \mathsf{occ} \lg\lg n)$	$O(1)$
Nishimoto et al. [17]	$O(z \lg n \lg^* n)$	$O(m \lg\lg n \lg\lg z + \lg z \lg m \lg n (\lg^* n)^2 + \mathsf{occ} \lg n)$	$n^{O(1)}$
Bille et al. [3]	$O(z \lg(n/z))$	$O(m + \mathsf{occ}(\lg^\epsilon n + \lg\lg n))$	$O(1)$
Bille et al. [3]	$O(z \lg(n/z) \lg\lg z)$	$O(m + \mathsf{occ} \lg\lg n)$	$n^{O(1)}$
Theorem 1	$O(z \lg(n/z))$	$O(m + \mathsf{occ}(\lg^\epsilon z + \lg\lg n))$	$O(1)$
Theorem 2 (1)	$O(z(\lg(n/z) + \lg\lg z))$	$O(m + \mathsf{occ}(\lg^\epsilon z + \lg\lg n))$	$n^{O(1)}$
Theorem 2 (2)	$O(z(\lg(n/z))$	$O(m + \mathsf{occ}(\lg^\epsilon z + \lg\lg n))^\dagger$	$n^{O(1)}$

1.1 Our Results

In this paper we improve previous solutions that are bounded by the size of the LZ77-parse. For constant-sized alphabets we obtain the following result:

Theorem 1. *Given a string S of length n from a constant-sized alphabet with an LZ77 parse of length z, we can build a compressed-index supporting pattern matching queries in $O(m + \mathsf{occ}(\lg\lg n + \lg^\epsilon z))$ time using $O(z \lg(n/z))$ space.*

In particular, we are the first to obtain optimal search time using only $O(z \lg(n/z))$ space. For general alphabets we obtain the following:

Theorem 2. *Given a string S of length n from an integer alphabet polynomially bounded by n with an LZ77-parse of length z, we can build a compressed-index supporting pattern matching queries in:*

(1) $O(m + \mathsf{occ}(\lg\lg n + \lg^\epsilon z))$ time using $O(z(\lg(n/z) + \lg\lg z))$ space.
(2) $O(m + \mathsf{occ}(\lg\lg n + \lg^\epsilon z))$ expected time using $O(z \lg(n/z))$ space.
(3) $O(m + \lg^\epsilon z + \mathsf{occ}(\lg\lg n + \lg^\epsilon z))$ time using $O(z \lg(n/z))$ space.

Note $\lg \lg z = O(\lg(n/z))$ when either the alphabet size is $O(2^{\lg^{\epsilon} n})$ or $z = o(\frac{n}{\lg^{\epsilon'} n})$ where ϵ and ϵ' are arbitrarily small positive constants. Theorem 1 follows directly from Theorem 2 (1) given these observations. Theorem 2 is a consequence of Lemmas 9, 11, 12 and 13.

1.2 Technical Overview

Our main new contribution is based on a new grammar construction. In [15] Melhorn et al. presented a way to maintain dynamic sequences subject to equality testing using a technique called signatures. They presented two signature construction techniques. One is randomized and leads to complexities that hold in expectation. The other is based on a deterministic coin-tossing technique of Cole and Vishkin [5] and leads to worst-case running times but incurs an iterated logarithmic overhead compared to the randomized solution. This technique has also resembles the string labeling techniques found e.g. in [19]. To the best of our knowledge, we are the first to consider grammar compression based on the randomized solution from [15]. Despite it being randomized we show how to obtain worst-case query bounds for text indexing using this technique.

The main idea in this grammar construction is that similar substrings will be parsed almost identically. This property also holds true for the deterministic construction technique which has been used to solve dynamic string problems with and without compression, see e.g. [1,17]. In [12] Jeż devices a different grammar construction algorithm with similar properties to solve the algorithmic pattern matching problem on grammar compressed strings which has later been used for both static and dynamic string problems, see [11,20]

Our primary solution has an $\lg^{\epsilon} z$ term in the query time which is problematic for short query patterns. To handle this, we show different solutions for handling short query patterns. These are based on the techniques from LZ77-based indexing combined with extra data structures to speed up the queries.

2 Preliminaries

We assume a standard unit-cost RAM model with word size $\Theta(\lg n)$ and that the input is from an integer alphabet $\Sigma = \{1, 2, \ldots, n^{O(1)}\}$. We measure space complexity in terms of machine words unless explicitly stated otherwise. A string S of length $n = |S|$ is a sequence of n symbols $S[1] \ldots S[n]$ drawn from an alphabet Σ. The sequence $S[i, j]$ is the *substring* of S given by $S[i] \ldots S[j]$ and strings can be concatenated, i.e. $S = S[1, k]S[k + 1, n]$. The empty string is denoted ϵ and $S[i, i] = S[i]$ while $S[i, j] = \epsilon$ if $j < i$, $S[i, j] = S[1, j]$ if $i < 1$ and $S[i, n]$ if $j > n$. The reverse of S denoted $rev(s)$ is the string $S[n]S[n-1] \ldots S[1]$. A *run* in a string S is a substring $S[i, j]$ with identical letters, i.e. $S[k] = S[k+1]$ for $k = i, \ldots, j-1$. Let $S[i, j]$ be a run in S then it is a *maximal run* if it cannot be extended, i.e. $S[i-1] \neq S[i]$ and $S[j] \neq S[j+1]$. If there are no runs in S we say that S is *run-free* and it follows that $S[i] \neq S[i+1]$ for $1 \leq i < n$. Denote by $[u]$ the set of integers $\{1, 2, \ldots, u\}$.

Let $X \subseteq [u]^2$ be a set of points in a 2-dimensional grid. The *2D-orthogonal range reporting problem* is to compactly represent Z while supporting *range reporting queries*, that is, given a rectangle $R = [a_1, b_1] \times [a_2, b_2]$ report all points in the set $R \cap X$. We use the following:

Lemma 1 (Chan et al. [4]). *For any set of n points in $[u] \times [u]$ and constant $\epsilon > 0$, we can solve 2D-orthogonal range reporting with $O(n \lg n)$ expected preprocessing time using:*

(i) $O(n)$ *space and* $(1 + k) \cdot O(\lg^{\epsilon} n \lg\lg u)$ *query time*
(ii) $O(n \lg\lg n)$ *space and* $(1 + k) \cdot O(\lg\lg u)$ *query time*

where k is the number of occurrences inside the rectangle.

A *Karp-Rabin fingerprinting function* [14] is a randomized hash function for strings. Given a string S of length n and a fingerprinting function ϕ we can in $O(n)$ time and space compute and store $O(n)$ fingerprints such that the fingerprint of any substring of S can be computed in constant time. Identical strings have identical fingerprints. The fingerprints of two strings S and S' *collide* when $S \neq S'$ and $\phi(S) = \phi(S')$. A fingerprinting function is *collision-free* for a set of strings when there are no collisions between the fingerprints of any two strings in the set. We can find collision-free fingerprinting function for a set of strings with total length n in $O(n)$ expected time [18].

Let D be a lexicographically sorted set of k strings. The weak prefix search problem is to compactly represent D while supporting *weak prefix queries*, that is, given a query string P of length m report the rank of the lexicographically smallest and largest strings in D of which P is a prefix. If no such strings exist, the answer can be arbitrary.

Lemma 2 (Belazzougui et al. [2], Appendix H.3). *Given a set D of k strings with average length l, from an alphabet of size σ, we can build a data structure using $O(k(\lg l + \lg\lg \sigma))$ bits of space supporting weak prefix search for a pattern P of length m in $O(m \lg \sigma / w + \lg m)$ time where w is the word size.*

We will refer to the data structure of Lemma 2 as a *z-fast trie* following the notation from [2]. The m term in the time complexity is due to a linear time preprocessing of the pattern and is not part of the actual search. Therefore it is simple to do weak prefix search for any length l substring of P in $O(\lg l)$ time after preprocessing P once in $O(m)$ time.

The *LZ77-parse* [21] of a string S of length n is a string $\mathcal{Z}$ of the form $(s_1, l_1, \alpha_1) \dots (s_z, l_z, \alpha_z) \in ([n], [n], \Sigma)^z$. We define $u_1 = 1$, $u_i = u_{i-1} + l_{i-1} + 1$ for $i > 1$. For $\mathcal{Z}$ to be a valid parse, we require $l_1 = 0$, $s_i < u_i$, $S[u_i, u_i + l_i - 1] = S[s_i, s_i + l_i - 1]$, and $S[u_i + l_i] = \alpha_i$ for $i \in [z]$. This guarantees $\mathcal{Z}$ *represents* S and S is uniquely defined in terms of $\mathcal{Z}$. The substring $S[u_i, u_i + l_i]$ is called the i^{th} phrase of the parse and $S[s_i, s_i + l_i - 1]$ is its source. A minimal LZ77-parse of S can be found greedily in $O(n)$ time and stored in $O(z)$ space [21]. We call the positions $u_1 + l_1, \dots, u_z + l_z$ the borders of S.

3 Signature Grammars

We consider a hierarchical representation of strings given by Melhorn et al. [15] with some slight modifications. Let S be a run-free string of length n from an integer alphabet Σ and let π be a uniformly random permutation of Σ. Define a position $S[i]$ as a local minimum of S if $1 < i < n$ and $\pi(S[i]) < \pi(S[i-1])$ and $\pi(S[i]) < \pi(S[i+1])$. In the block decomposition of S, a block starts at position 1 and at every local minimum in S and ends just before the next block begins (the last block ends at position n). The block decomposition of a string S can be used to construct the signature tree of S denoted $sig(S)$ which is an ordered labeled tree with several useful properties.

Lemma 3. *Let S be a run-free string S of length n from an alphabet Σ and let π be a uniformly random permutation of Σ such that $\pi(c)$ is the rank of the symbol $c \in \Sigma$ in this permutation. Then the expected length between two local minima in the sequence $\pi(S[1]), \pi(S[2]), \ldots, \pi(S[n])$ is at most 3 and the longest gap is $O(\lg n)$ in expectation.*

Proof. First we show the expected length between two local minima is at most 3. Look at a position $1 \leq i \leq n$ in the sequence $\pi(S[1]), \pi(S[2]), \ldots, \pi(S[n])$. To determine if $\pi(S[i])$ is a local minimum, we only need to consider the two neighbouring elements $\pi(S[i-1])$ and $\pi(S[i+1])$ thus let us consider the triple $(\pi(S[i-1]), \pi(S[i]), \pi(S[i+1]))$. We need to consider the following cases. First assume $S[i-1] \neq S[i] \neq S[i+1]$. There exist $3! = 6$ permutations of a triple with unique elements and in two of these the minimum element is in the middle. Since π is a uniformly random permutation of Σ all 6 permutations are equally likely, and thus there is 1/3 chance that the element at position i is a local minimum. Now instead assume $S[i-1] = S[i+1] \neq S[i]$ in which case there is 1/2 chance that the middle element is the smallest. Finally, in the case where $i = 1$ or $i = n$ there is also 1/2 chance. As S is run-free, these cases cover all possible cases. Thus there is at least 1/3 chance that any position i is a local minimum independently of S. Thus the expected number of local minima in the sequence is therefore at least $n/3$ and the expected distance between any two local minima is at most 3.

The expected longest distance between two local minima of $O(\lg n)$ was shown in [15].

3.1 Signature Grammar Construction

We now give the construction algorithm for the signature tree $sig(S)$. Consider an ordered forest F of trees. Initially, F consists of n trees where the i^{th} tree is a single node with label $S[i]$. Let the label of a tree t denoted $l(t)$ be the label of its root node. Let $l(F)$ denote the string that is given by the in-order concatenation of the labels of the trees in F. The construction of $sig(S)$ proceeds as follows:

1. Let $t_i, \ldots, t_j$ be a maximal subrange of consecutive trees of F with identical labels, i.e. $l(t_i) = \ldots = l(t_j)$. Replace each such subrange in F by a new tree

having as root a new node v with children $t_i, \ldots, t_j$ and a label that identifies the number of children and their label. We call this kind of node a run node. Now $l(F)$ is run-free.
2. Consider the block decomposition of $l(F)$. Let $t_i, \ldots, t_j$ be consecutive trees in F such that their labels form a block in $l(F)$. Replace all identical blocks $t_i, \ldots, t_j$ by a new tree having as root a new node with children $t_i, \ldots, t_j$ and a unique label. We call this kind of node a run-free node.
3. Repeat steps 1 and 2 until F contains a single tree, we call this tree $sig(S)$.

In each iteration the size of F decreases by at least a factor of two and each iteration takes $O(|F|)$ time, thus it can be constructed in $O(n)$ time.

Consider the directed acyclic graph (DAG) of the tree $sig(S)$ where all identical subtrees are merged. Note we can store run nodes in $O(1)$ space since all out-going edges are pointing to the same node, so we store the number of edges along with a single edge instead of explicitly storing each of them. For run-free nodes we use space proportional to their out-degrees. We call this the signature DAG of S denoted $dag(S)$. There is a one-to-one correspondence between this DAG and an acyclic run-length grammar producing S where each node corresponds to a production and each leaf to a terminal.

3.2 Properties of the Signature Grammar

We now show some properties of $sig(S)$ and $dag(S)$ that we will need later. Let $str(v)$ denote the substring of S given by the labels of the leaves of the subtree of $sig(S)$ induced by the node v in left to right order.

Lemma 4. *Let v be a node in the signature tree for a string S of length n. If v has height h then $|str(v)|$ is at least 2^h and thus $sig(S)$ (and $dag(S)$) has height $O(\lg n)$.*

Proof. This follows directly from the out-degree of all nodes being at least 2.

Denote by $T(i,j)$ the set of nodes in $sig(S)$ that are ancestors of the i^{th} through j^{th} leaf of $sig(S)$. These nodes form a sequence of adjacent nodes at every level of $sig(S)$ and we call them *relevant nodes* for the substring $S[i,j]$.

Lemma 5. *$T(i,j)$ and $T(i',j')$ have identical nodes except at most the two first and two last nodes on each level whenever $S[i,j] = S[i',j']$.*

Proof. Trivially, the leaves of $T(i,j)$ and $T(i',j')$ are identical if $S[i,j] = S[i',j']$. Now we show it is true for nodes on level l assuming it is true for nodes on level $l-1$. We only consider the left part of each level as the argument for the right part is (almost) symmetric. Let $v_1, v_2, v_3, \ldots$ be the nodes on level $l-1$ in $T(i,j)$ and $u_1, u_2, u_3, \ldots$ the nodes on level $l-1$ in $T(i',j')$ in left to right order. From the assumption, we have $v_a, v_{a+1}, \ldots$ are identical with $u_b, u_{b+1}, \ldots$ for some $1 \leq a, b \leq 3$. When constructing the l^{th} level of $sig(S)$, these nodes are divided into blocks. Let v_{a+k} be the first block that starts after v_a then by the block decomposition, the first block after u_b starts at u_{b+k}. The nodes $v_1, \ldots, v_{a+k}$ are spanned by at most two blocks and similarly for $u_1, \ldots, u_{b+k}$. These blocks become the first one or two nodes on level l in $T(i,j)$ and $T(i',j')$ respectively. The block starting at v_{a+k} is identical to the block starting at u_{b+k} and the same holds for the following blocks. These blocks result in identical nodes on level l. Thus, if we ignore the at most two first (and last) nodes on level l the remaining nodes are identical.

We call nodes of $T(i,j)$ consistent in respect to $T(i,j)$ if they are guaranteed to be in any other $T(i',j')$ where $S[i,j] = S[i',j']$. We denote the remaining nodes of $T(i,j)$ as inconsistent. From the above lemma, it follows at most the left-most and right-most two nodes on each level of $T(i,j)$ can be inconsistent.

Lemma 6. *The expected size of the signature DAG dag*(S) *is* $O(z \lg(n/z))$.

Proof. We first bound the number of unique nodes in *sig*(S) in terms of the LZ77-parse of S which has size z. Consider the decomposition of S into the $2z$ substrings $S[u_1, u_1+l_1], S[u_1+l_1+1], \ldots, S[u_z, u_z+l_z], S[u_z+l_z+1]$ given by the phrases and borders of the LZ77-parse of S and the corresponding sets of relevant nodes $R = \{T(u_1, u_1+l_1), T(u_1+l_1+1, u_1+l_1+1), \ldots\}$. Clearly, the union of these sets are all the nodes of *sig*(S). Since identical nodes are represented only once in *dag*(S) we need only count one of their occurrences in *sig*(S). We first count the nodes at levels lower than $\lg(n/z)$. A set $T(i,i)$ of nodes relevant to a substring of length one has no more than $O(\lg(n/z))$ such nodes. By Lemma 5 only $O(\lg(n/z))$ of the relevant nodes for a phrase are not guaranteed to also appear in the relevant nodes of its source. Thus we count a total of $O(z \lg(n/z))$ nodes for the $O(z)$ sets of relevant nodes. Consider the leftmost appearance of a node appearing one or more times in *sig*(S). By definition, and because every node of *sig*(S) is in at least one relevant set, it must already be counted towards one of the sets. Thus there are $O(z \lg(n/z))$ unique vertices in *sig*(S) at levels lower than $\lg(n/z)$. Now for the remaining at most $\lg(z)$ levels, there are no more than $O(z)$ nodes because the out-degree of every node is at least two. Thus we have proved that there are $O(z \lg(n/z))$ unique nodes in *sig*(S). By Lemma 3 the average block size and thus the expected out-degree of a node is $O(1)$. It follows that the expected number of edges and the expected size of *dag*(S) is $O(z \lg(n/z))$.

Lemma 7. *A signature grammar of S using* $O(z \lg(n/z))$ *(worst case) space can be constructed in* $O(n)$ *expected time.*

Proof. Construct a signature grammar for S using the signature grammar construction algorithm. If the average out-degree of the run-free nodes in *dag*(S) is more than some constant greater than 3 then try again. In expectation it only takes a constant number of retries before this is not the case.

Lemma 8. *Given a node* $v \in$ *dag*(S)*, the child that produces the character at position i in str*(v) *can be found in* $O(1)$ *time.*

Proof. First assume v is a run-free node. If we store $|str(u)|$ for each child u of v in order, the correct child corresponding to position i can simply be found by iterating over these. However, this may take $O(\log n)$ time since this is the maximum out-degree of a node in $dag(S)$. This can be improved to $O(\log\log n)$ by doing a binary search, but instead we use a Fusion Tree from [7] that allows us to do this in $O(1)$ time since we have at most $O(\log n)$ elements. This does not increase the space usage. If v is a run node then it is easy to calculate the right child by a single division.

4 Long Patterns

In this section we present how to use the signature grammar to construct a compressed index that we will use for patterns of length $\Omega(\lg^{\epsilon} z)$ for constant $\epsilon > 0$. We obtain the following lemma:

Lemma 9. *Given a string S of length n with an LZ77-parse of length z we can build a compressed index supporting pattern matching queries in $O(m + (1 + \mathsf{occ}) \lg^{\epsilon} z)$ time using $O(z \lg(n/z))$ space for any constant $\epsilon > 0$.*

4.1 Data Structure

Consider a vertex v with children $u_1, \ldots, u_k$ in $dag(S)$. Let $pre(v, i)$ denote the prefix of $str(v)$ given by concatenating the strings represented by the first i children of v and let $suf(v, i)$ be the suffix of $str(v)$ given by concatenating the strings represented by the last $k - i$ children of x.

The data structure is composed of two z-fast tries (see Lemma 2) T_1 and T_2 and a 2D-range reporting data structure R.

For every non-leaf node $v \in dag(S)$ we store the following. Let k be the number of children of v if v is a run-free node otherwise let $k = 2$:

- The reverse of the strings $pre(v, i)$ for $i \in [k - 1]$ in the z-fast trie T_1.
- The strings $suf(v, i)$ for $i \in [k - 1]$ in the z-fast trie T_2.
- The points (a, b) where a is the rank of the reverse of $pre(v, i)$ in T_1 and b is the rank of $suf(v, i)$ in T_2 for $i \in [k - 1]$ are stored in R. A point stores the vertex $v \in dag(S)$ and the length of $pre(v, i)$ as auxiliary information.

There are $O(z \lg(n/z))$ vertices in $dag(S)$ thus T_1 and T_2 take no more than $O(z \lg(n/z))$ words of space using Lemma 2. There $O(z \lg(n/z))$ points in R which takes $O(z \lg(n/z))$ space using Lemma 1 (i) thus the total space in words is $O(z \lg(n/z))$.

4.2 Searching

Assume in the following that there are no fingerprint collisions. Compute all the prefix fingerprints of P $\phi(P[1]), \phi(P[1, 2]), \ldots, \phi(P[1, m])$. Consider the signature tree $sig(P)$ for P. Let l_i^k denote the k'th left-most vertex on level i in $sig(P)$ and let j be the last level. Let $P_L = \{|str(l_1^1)|, |str(l_1^1)| + |str(l_1^2)|, |str(l_2^1)|, |str(l_2^1)| + |str(l_2^2)|, \ldots, |str(l_j^1)|, |str(l_j^1)| + |str(l_j^2)|\}$. Symmetrically, let r_i^k denote the k'th right-most vertex on level i in $sig(P)$ and let $P_R = \{m - |str(r_1^1)|, m - |str(r_1^1)| - |str(r_1^2)|, m - |str(r_2^1)|, m - |str(r_2^1)| - |str(r_2^2)|, \ldots, m - |str(r_j^1)|, m - |str(r_j^1)| - |str(r_j^2)|\}$. Let $P_S = P_L \cup P_R$.

For $p \in P_S$ search for the reverse of $P[1, p]$ in T_1 and for $P[p+1, m]$ in T_2 using the precomputed fingerprints. Let $[a, b]$ and $[c, d]$ be the respective ranges returned by the search. Do a range reporting query for the (possibly empty) range $[a, b] \times [c, d]$ in R. Each point in the range identifies a node v and a position i such that P occurs at position i in the string $str(v)$. If v is a run node, there is furthermore an occurrence of P in $str(v)$ for all positions $i + k \cdot |str(child(v))|$ where $k = 1, \ldots, j$ and $j \cdot |str(child(v))| + m \leq str(v)$.

To report the actual occurrences of P in S we traverse all ancestors of v in $dag(S)$; for each occurrence of P in $str(v)$ found, recursively visit each parent u of v and offset the location of the occurrence to match the location in $str(u)$ instead of $str(v)$. When u is the root, report the occurrence. Observe that the time it takes to traverse the ancestors of v is linear in the number of occurrences we find.

We now describe how to handle fingerprint collisions. Given a z-fast trie, Gagie et al. [9] show how to perform k weak prefix queries and identify all false positives using $O(k \lg m + m)$ extra time by employing bookmarked extraction and bookmarked fingerprinting. Because we only compute fingerprints and extract prefixes (suffixes) of the strings represented by vertices in $dag(S)$ we do not need bookmarking to do this. We refer the reader to [9] for the details. Thus, we modify the search algorithm such that all the searches in T_1 and T_2 are carried out first, then we verify the results before progressing to doing range reporting queries only for ranges that were not discarded during verification.

4.3 Correctness

For any occurrence $S[l, r]$ of P in S there is a node v in $sig(S)$ that stabs $S[l, r]$, i.e. a suffix of $pre(v, i)$ equals a prefix $P[1, j]$ and a prefix of $suf(v, i)$ equals the remaining suffix $P[j+1, m]$ for some i and j. Since we put all combinations of $pre(v, i)$, $suf(v, i)$ into T_1, T_2 and R, we would be guaranteed to find all nodes v that contains P in $str(v)$ if we searched for all possible split-points $1, \ldots, m-1$ of P i.e. $P[1, i]$ and $P[i+1, m]$ for $i = 1, \ldots, m-1$.

We now argue that we do not need to search for all possible split-points of P but only need to consider those in the set P_S. For a position i, we say the node v stabs i if the nearest common ancestor of the i^{th} and $i+1^{th}$ leaf of $sig(S)$ denoted $NCA(l_i, l_{i+1})$ is v.

Look at any occurrence $S[l, r]$ of P. Consider $T_S = T(l, r)$ and $T_P = sig(P)$. Look at a possible split-point $i \in [1, m-1]$ and the node v that stabs position i in T_P. Let u_l and u_r be adjacent children of v such that the rightmost leaf descendant of u_l is the i^{th} leaf and the leftmost leaf descendant of u_r is the $i+1^{th}$ leaf. We now look at two cases for v and argue it is irrelevant to consider position i as split-point for P in these cases:

1. **Case v is consistent (in respect to T_P).** In this case it is guaranteed that the node that stabs $l+i$ in T_S is identical to v. Since v is a descendant of the root of T_P (as the root of T_P is inconsistent) $str(v)$ cannot contain P and thus it is irrelevant to consider i as a split-point.
2. **Case v is inconsistent and u_l and u_r are both consistent (in respect to T_P).** In this case u_l and u_r have identical corresponding nodes u'_l and u'_r in T_S. Because u_l and u_r are children of the same node it follows that u'_l and u'_r must also both be children of some node v' that stabs $l+i$ in T_S (however v and v' may not be identical since v is inconsistent). Consider the node u'_{ll} to the left of u'_l (or symmetrically for the right side if v is an inconsistent node in the right side of T_P). If $str(v')$ contains P then u'_{ll} is also a child of v' (otherwise u_l would be inconsistent). So it suffices to check the split-point $i-|u_l|$. Surely $i-|u_l|$ stabs an inconsistent node in T_P, so either we consider that position relevant, or the same argument applies again and a split-point further to the left is eventually considered relevant.

Thus only split-points where v and at least one of u_l or u_r are inconsistent are relevant. These positions are a subset of the position in P_S, and thus we try all relevant split-points.

4.4 Complexity

A query on T_1 and T_2 takes $O(\lg m)$ time by Lemma 2 while a query on R takes $O(\lg^{\epsilon} z)$ time using Lemma 1 (i) (excluding reporting). We do $O(\lg m)$ queries as the size of P_S is $O(\lg m)$. Verification of the $O(\lg m)$ strings we search for takes total time $O(\lg^2 m + m) = O(m)$. Constructing the signature DAG for P takes $O(m)$ time, thus total time without reporting is $O(m+\lg m \lg^{\epsilon} z) = O(m+\lg^{\epsilon'} z)$ for any $\epsilon' > \epsilon$. This holds because if $m \leq \lg^{2\epsilon} z$ then $\lg m \lg^{\epsilon} z \leq \lg\lg^{2\epsilon} z \lg^{\epsilon} z = O(\lg^{\epsilon'} z)$, otherwise $m > \lg^{2\epsilon} z \Leftrightarrow \sqrt{m} > \lg^{\epsilon} z$ and then $\lg m \lg^{\epsilon} z = O(\lg m\sqrt{m}) = O(m)$. For every query on R we may find multiple points each corresponding to an occurrence of P. It takes $O(\lg^{\epsilon} z)$ time to report each point thus the total time becomes $O(m + (1+\mathsf{occ})\lg^{\epsilon'} z)$.

5 Short Patterns

Our solution for short patterns uses properties of the LZ77-parse of S. A *primary* substring of S is a substring that contains one or more borders of S, all other substrings are called *secondary.* A primary substring that matches a query pattern P is a *primary occurrence* of P while a secondary substring that matches P is a *secondary occurrence* of P. In a seminal paper on LZ77 based indexing [13] Kärkkäinen and Ukkonen use some observations by Farach and Thorup [6] to show how all secondary occurrences of a query pattern P can be found given a list of the primary occurrences of P through a reduction to orthogonal range reporting. Employing the range reporting result given in Lemma 1 (ii), all secondary occurrences can be reported as stated in the following lemma:

Lemma 10 (Kärkkäinen and Ukkonen [13]). *Given the LZ77-parse of a string S there exists a data structure that uses $O(z \lg\lg z)$ space that can report all secondary occurrences of a pattern P given the list of primary occurrences of P in S in $O(\mathsf{occ} \lg\lg n)$ time.*

We now describe a data structure that can report all primary occurrences of a pattern P of length at most k in $O(m + \mathsf{occ})$ time using $O(zk)$ space.

Lemma 11. *Given a string S of length n and a positive integer $k \leq n$ we can build a compressed index supporting pattern matching queries for patterns of length m in $O(m + \mathsf{occ} \lg\lg n)$ time using $O(zk + z \lg\lg z)$ space that works for $m \leq k$.*

Proof. Consider the set C of z substrings of S that are defined by $S[u_i - k, u_i + k - 1]$ for $i \in [z]$, i.e. the substrings of length $2k$ surrounding the borders of the LZ77-parse. The total length of these strings is $\Theta(zk)$. Construct the generalized suffix tree T over the set of strings C. This takes $\Theta(zk)$ words of space. To ensure no occurrence is reported more than once, if multiple suffixes in this generalized suffix tree correspond to substrings of S that starts on the same position in S, only include the longest of these. This happens when the distance between two borders is less than $2k$.

To find the primary occurrences of P of length m, simply find all occurrences of P in T. These occurrences are a super set of the primary occurrences of P in S, since T contains all substrings starting/ending at most k positions from a border. It is easy to filter out all occurrences that are not primary, simply by calculating if they cross a border or not. This takes $O(m + \mathsf{occ})$ time (where occ includes secondary occurrences). Combined with Lemma 10 this gives Lemma 11.

6 Semi-short Patterns

In this section, we show how to handle patterns of length between $\lg\lg z$ and $\lg^\epsilon z$. It is based on the same reduction to 2D-range reporting as used for long patterns. However, the positions in S that are inserted in the range reporting structure is now based on the LZ77-parse of S instead. Furthermore we use Lemma 1 (ii) which gives faster range reporting but uses super-linear space, which is fine because we instead put fewer points into the structure. We get the following lemma:

Lemma 12. *Given a string S of length n we solve the compressed indexing problem for a pattern P of length m with $\lg\lg z \leq m \leq \lg^\epsilon z$ for any positive constant $\epsilon < \frac{1}{2}$ in $O(m + \mathsf{occ}(\lg\lg n + \lg^\epsilon z))$ time using $O(z(\lg\lg z + \log(n/z)))$ space.*

6.1 Data Structure

As in the previous section for short patterns, we only need to worry about primary occurrences of P in S. Let B be the set of all substrings of length at most $\lg^{\epsilon} z$ that cross a border in S. The split positions of such a string are the offsets of the leftmost borders in its occurrences. All primary occurrences of P in S are in this set. The size of this set is $|B| = O(z \lg^{2\epsilon} z)$. The data structure is composed by the following:

- A dictionary H mapping each string in B to its split positions.
- A z-fast trie T_1 on the reverse of the strings $T[u_i, l_i]$ for $i \in [z]$.
- A z-fast trie T_2 on the strings $T[u_i, n]$ for $i \in [z]$.
- A range reporting data structure R with a point (c, d) for every pair of strings $C_i = T[u_i, l_i], D_i = T[u_{i+1}, n]$ for $i \in [z]$ where $D_z = \epsilon$ and c is the lexicographical rank of the reverse of C_i in the set $\{C_1, \ldots, C_z\}$ and d is the lexicographical rank of D_i in the set $\{D_1, \ldots D_z\}$. We store the border u_i along with the point (c, d).
- The data structure described in Lemma 10 to report secondary occurrences.
- The signature grammar for S.

Each entry in H requires $\lg \lg^{\epsilon} z = O(\lg \lg z)$ bits to store since a split position can be at most $\lg^{\epsilon} z$. Thus the dictionary can be stored in $O(|B| \cdot \lg \lg z) = O(z \lg^{2\epsilon} z \lg \lg z)$ bits which for $\epsilon < \frac{1}{2}$ is $O(z)$ words. The tries T_1 and T_2 take $O(z)$ space while R takes $O(z \lg \lg z)$ space. The signature grammar takes $O(z \log(n/z))$. Thus the total space is $O(z(\lg \lg z + \log(n/z)))$.

6.2 Searching

Assume a lookup for P in H does not give false-positives. Given a pattern P compute all prefix fingerprints of P. Next do a lookup in H. If there is no match then P does not occur in S. Otherwise, we do the following for each of the split-points s stored in H. First split P into a left part $P_l = P[0, s-1]$ and a right part $P_r = P[s, m]$. Then search for the reverse of P_l in T_1 and for P_r in T_2 using the corresponding fingerprints. The search induces a (possibly empty) range for which we do a range reporting query in R. Each occurrence in R corresponds to a primary occurrence of P in S, so report these. Finally use Lemma 10 to report all secondary occurrences.

Unfortunately, we cannot guarantee a lookup for P in H does not give a false positive. Instead, we pause the reporting step when the first possible occurrence of P has been found. At this point, we verify the substring P matches the found occurrence in S. We know this occurrence is around an LZ-border in S such that P_l is to the left of the border and P_r is to the right of the border. Thus we can efficiently verify that P actually occurs at this position using the grammar.

6.3 Analysis

Computing the prefix fingerprints of P takes $O(m)$ time. First, we analyze the running time in the case P actually exists in S. The lookup in H takes $O(1)$ time using perfect hashing. For each split-point we do two z-fast trie lookups in time $O(\lg m) = O(\lg \lg z)$. Since each different split-point corresponds to at least one unique occurrence, this takes at most $O(\mathsf{occ} \lg\lg z)$ time in total. Similarly each lookup and occurrence in the 2D-range reporting structure takes $\lg\lg z$ time, which is therefore also bounded by $O(\mathsf{occ} \lg\lg z)$ time. Finally, we verified one of the found occurrence against P in $O(m)$ time. So the total time is $O(m + \mathsf{occ} \lg\lg z)$ in this case.

In the case P does not exists, either the lookup in H tells us that, and we spend $O(1)$ time, or the lookup in H is a false-positive. In the latter case, we perform exactly two z-fast trie lookups and one range reporting query. These all take time $O(\lg\lg z)$. Since $m \geq \lg\lg z$ this is $O(m)$ time. Again, we verified the found occurrence against P in $O(m)$ time. The total time in this case is therefore $O(m)$.

Note we ensure our fingerprint function is collision free for all substrings in B during the preprocessing thus there can only be collisions if P does not occur in S when $m \leq \lg^\epsilon z$.

7 Randomized Solution

In this section we present a very simple way to turn the $O(m + (1 + \mathsf{occ}) \lg^\epsilon z)$ worst-case time of Lemma 9 into $O(m + \mathsf{occ} \lg^\epsilon z)$ expected time. First observe, this is already true if the pattern we search for occurs at least once or if $m \geq \lg^\epsilon z$.

As in the semi-short patterns section, we consider the set B of substrings of S of length at most $\lg^\epsilon z$ that crosses a border. Create a dictionary H with $z \lg^{3\epsilon} z$ entries and insert all the strings from B. This means only a $\lg^\epsilon z$ fraction of the entries are used, and thus if we lookup a string s (where $|s| \leq \lg^\epsilon z$) that is not in H there is only a $\frac{1}{\lg^\epsilon z}$ chance of getting a false-positive.

Now to answer a query, we first check if $m \leq \lg^\epsilon z$ in which case we look it up in H. If it does not exist, report that. If it does exist in H or if $m > \lg^\epsilon z$ use the solution from Lemma 9 to answer the query.

In the case P does not exist, we spend either $O(m)$ time if H reports no, or $O(m + \lg^\epsilon z)$ time if H reports a false-positive. Since there is only $\frac{1}{\lg^\epsilon z}$ chance of getting a false positive, the expected time in this case is $O(m)$. In all other cases, the running time is $O(m + \mathsf{occ} \lg^\epsilon z)$ in worst-case, so the total expected running time is $O(m + \mathsf{occ} \lg^\epsilon z)$. The space usage of H is $O(z \lg^{3\epsilon} z)$ bits since we only need to store one bit for each entry. This is $O(z)$ words for $\epsilon \leq 1/3$. To sum up, we get the following lemma:

Lemma 13. *Given a signature grammar for a text S of length n with an LZ77-parse of length z we can build a compressed index supporting pattern matching queries in $O(m + \mathsf{occ} \lg^\epsilon z)$ expected time using $O(z \lg(n/z))$ space for any constant $0 < \epsilon \leq 1/3$.*

References

1. Alstrup, S., Brodal, G.S., Rauhe, T.: Pattern matching in dynamic texts. In: Proceedings of the 11th Annual Symposium on Discrete Algorithms. Citeseer (2000)
2. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Fast prefix search in little space, with applications. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 427–438. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15775-2_37
3. Bille, P., Ettienne, M.B., Gørtz, I.L., Vildhøj, H.W.: Time-space trade-offs for Lempel-Ziv compressed indexing. In: 28th Annual Symposium on Combinatorial Pattern Matching. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2017)
4. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Proceedings of the 27th SOCG, pp. 1–10 (2011)
5. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**(1), 32–53 (1986)
6. Farach, M., Thorup, M.: String matching in Lempel-Ziv compressed strings. Algorithmica **20**(4), 388–404 (1998)
7. Fredman, M.L., Willard, D.E.: Blasting through the information theoretic barrier with fusion trees. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC 1990, pp. 1–7. ACM, New York (1990)
8. Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: A faster grammar-based self-index. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 240–251. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28332-1_21
9. Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: LZ77-based self-indexing with faster pattern matching. In: Pardo, A., Viola, A. (eds.) LATIN 2014. LNCS, vol. 8392, pp. 731–742. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54423-1_63
10. Gagie, T., Navarro, G., Prezza, N.: Optimal-time text indexing in BWT-runs bounded space. arXiv preprint arXiv:1705.10382 (2017)
11. Gawrychowski, P., Karczmarz, A., Kociumaka, T., Łącki, J., Sankowski, P.: Optimal dynamic strings. arXiv preprint arXiv:1511.02612 (2015)
12. Jeż, A.: Faster fully compressed pattern matching by recompression. ACM Trans. Algorithms (TALG) **11**(3), 20 (2015)
13. Kärkkäinen, J., Ukkonen, E.: Lempel-Ziv parsing and sublinear-size index structures for string matching. In: Proceedings of the 3rd South American Workshop on String Processing (WSP 1996), vol. 26, no. (Teollisuuskatu 23), pp. 141–155 (1996)
14. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. **31**(2), 249–260 (1987)
15. Mehlhorn, K., Sundar, R., Uhrig, C.: Maintaining dynamic sequences under equality tests in polylogarithmic time. Algorithmica **17**(2), 183–198 (1997)
16. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. (CSUR) **39**(1), 2 (2007)
17. Nishimoto, T., Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Dynamic index, LZ factorization, and LCE queries in compressed space. arXiv preprint arXiv:1504.06954 (2015)
18. Porat, B., Porat, E.: Exact and approximate pattern matching in the streaming model. In: Proceedings of the 50th FOCS, pp. 315–323 (2009)

19. Sahinalp, S.C., Vishkin, U.: Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proceedings of 37th Conference on Foundations of Computer Science, October 1996
20. Tomohiro, I.: Longest common extension with recompression (2017)
21. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977)

Combinatorics of Beacon-Based Routing in Three Dimensions

Jonas Cleve(✉) and Wolfgang Mulzer

Institut für Informatik, Freie Universität Berlin, Berlin, Germany
{jonascleve,mulzer}@inf.fu-berlin.de

Abstract. A beacon is a point-like object which can be enabled to exert a magnetic pull on other point-like objects in space. Those objects then move towards the beacon in a greedy fashion until they are either stuck at an obstacle or reach the beacon's location. Beacons placed inside polyhedra can be used to route point-like objects from one location to another. A second use case is to cover a polyhedron such that every point-like object at an arbitrary location in the polyhedron can reach at least one of the beacons once the latter is activated.

The notion of beacon-based routing and guarding was introduced by Biro et al. [FWCG'11] in 2011 and covered in detail by Biro in his Ph.D. thesis [SUNY-SB'13], which focuses on the two-dimensional case.

We extend Biro's result to three dimensions by considering beacon routing in polyhedra. We show that $\lfloor \frac{m+1}{3} \rfloor$ beacons are always sufficient and sometimes necessary to route between any pair of points in a given polyhedron P, where m is the number of tetrahedra in a tetrahedral decomposition of P. This is one of the first results that show that beacon routing is also possible in three dimensions.

1 Introduction

A *beacon* b is a point-like object in a polyhedron P which can be enabled to exert a magnetic pull on all points inside P. Those points then move in the direction in which the distance to b decreases most rapidly. As long as the distance decreases, points can also move along obstacles they hit on their way.

The resulting *attraction path* alternates between unrestricted movement inside P and restricted movement on the boundary of P. If the attraction path of a point p towards a beacon b ends in b we say that b *covers* p. On the other hand, p is *stuck* if it is in a position where it cannot decrease its distance to b.

A point p can be *routed via beacons* towards a point q if there exists a sequence of beacons $b_1, b_2, \ldots, b_k = q$ such that b_1 covers p and b_{i+1} covers b_i for all $1 \leq i < k$. In our model at most one beacon can be enabled at any time and a point has to reach the beacon's location before the next beacon can be enabled.

Supported in part by DFG grant MU 3501/1 and ERC StG 757609.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 346–360, 2018.
https://doi.org/10.1007/978-3-319-77404-6_26

The notion of beacon attraction was introduced by Biro et al. [4,5] for two dimensions. This extends the classic notion of visibility [9]: the visibility region of a point is a subset of the attraction region of a point.

Here, we study the case of three-dimensional polyhedra. A three-dimensional polytope or *polyhedron* is a compact connected set bounded by a piecewise linear 2-manifold. The results in this work are based on the master's thesis of the first author [8] in which various aspects of beacon-based routing and guarding were studied in three dimensions. Simultaneously, Aldana-Galván et al. [1,2] looked at orthogonal polyhedra and introduced the notion of *edge beacons*.

For two dimensions, Biro [4] provided bounds on the number of beacons for routing in a polygon. He also showed that it is NP-hard and APX-hard to find a minimum set of beacons for a given polygon such that it is possible to (a) route between any pair of points, (b) route one specific source point to any other point, (c) route any point to one specific target point, or (d) cover the polygon.

It is easy to reduce the two-dimensional problems to their three-dimensional counterparts by lifting the polygon into three dimensions. It thus follows that the corresponding problems in three dimensions are also NP-hard and APX-hard. More details can be found in [8, Chap. 4].

2 Preliminary Thoughts on Tetrahedral Decompositions

To show an upper bound on the number of beacons necessary to route between any pair of points in two dimensions Biro et al. [5] look at a triangulation of the polygon. They show that for every two additional triangles at most one beacon is needed. Even though there is a slight flaw in the case analysis of their proof, this can be easily repaired, see [8, Chap. 3.1] for more details and the working proof. We extend this approach to three dimensions by looking at the decomposition of a polyhedron into tetrahedra.

The tetrahedral decomposition is no general solution: Lennes [10] has shown in 1911 that a polyhedron cannot, in general, be decomposed into tetrahedra if no additional vertices are allowed. The problem of deciding whether such a decomposition exists is, in fact, NP-complete as shown by Ruppert and Seidel [11].

In the two-dimensional case, every simple polygon with n vertices has a triangulation with exactly $n-2$ triangles; a polygon with h holes has a triangulation of $n-2+2h$ triangles. In contrast, for three dimensions the number of tetrahedra in a tetrahedral decomposition is not directly related to the number of vertices. Chazelle [7] showed that for arbitrary n there exists a polyhedron with $\Theta(n)$ vertices for which at least $\Omega(n^2)$ convex parts are needed to decompose it. Naturally, this is also a worst-case lower bound on the number of tetrahedra. On the other hand, Bern and Eppstein [3, Theorem 13] show that any polyhedron can be triangulated with $\mathcal{O}(n^2)$ tetrahedra with the help of $\mathcal{O}(n^2)$ *Steiner points*. Furthermore, it is clear that every tetrahedral decomposition consists of at least $n-3$ tetrahedra.

One polyhedron can have different tetrahedral decompositions with different numbers of tetrahedra. An example of such a polyhedron is a triangular bipyramid which can be decomposed into two or three tetrahedra, see [11, p. 228].

Due to this, we will prove bounds on the number of beacons needed for routing relative to the number of tetrahedra m rather than the number of vertices n. Since we accept any kind of decomposition and do not have any general position assumption tetrahedral decompositions with Steiner points are allowed.

To successfully apply the ideas for two dimensions to three dimensions we need the following preliminary definition and lemma.

Definition 2.1 (Dual graph of tetrahedral decompositions). *Given a polyhedron with a tetrahedral decomposition $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ into m tetrahedra, its* dual graph *is an undirected graph $D(\Sigma) = (V, E)$ where*

(i) $V = \{\sigma_1, \ldots, \sigma_m\}$ and
(ii) $E = \{\{\sigma_i, \sigma_j\} \in \binom{V}{2} \mid \sigma_i$ and σ_j share exactly one triangular facet$\}$.

Observation 2.2. Unlike in two dimensions, the dual graph of a tetrahedral decomposition is not necessarily a tree. We can still observe that each node in the dual graph has at most 4 neighbors—one for each facet of the tetrahedron.

Lemma 2.3. *Given a tetrahedral decomposition Σ of a polyhedron together with its dual graph $D(\Sigma)$ and a subset $S \subseteq \Sigma$ of tetrahedra from the decomposition whose induced subgraph $D(S)$ of $D(\Sigma)$ is connected, then*

(i) $|S| = 2$ implies that the tetrahedra in S share one triangular facet,
(ii) $|S| = 3$ implies that the tetrahedra in S share one edge, and
(iii) $|S| = 4$ implies that the tetrahedra in S share at least one vertex.

Proof. We show this seperately for every case.

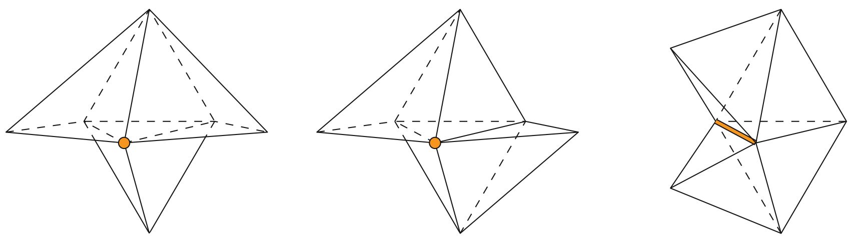
(i) This follows directly from Definition 2.1.
(ii) In a connected graph of three nodes there is one node neighboring the other two. By Definition 2.1 the dual tetrahedron shares one facet with each of the other tetrahedra. In a tetrahedron every pair of facets shares one edge.
(iii) By case (ii) there is a subset of three (connected) tetrahedra that shares one edge e. This edge is therefore part of each of the three tetrahedra. By Definition 2.1, the fourth tetrahedron shares a facet f with at least one of the other three (called σ). Since f contains three and e two vertices of σ they share at least one vertex. A depiction of the possible configurations of four tetrahedra can be seen in Fig. 1. □

3 An Upper Bound for Beacon-Based Routing

After the preparatory work, we can now show an upper bound on the number of beacons needed to route within a polyhedron with a tetrahedral decomposition. The idea of the proof is based on the proof by Biro et al. [5] for (two-dimensional) polygons. We want to show the following

Hypothesis 3.1. Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.

Since the proof is quite long and consists of many cases it is split up into various lemmas which are finally combined in Theorem 3.7.


(a) One tetrahedron in the center has all other tetrahedra as neighbors.

(b) Two tetrahedra with one and two tetrahedra with two neighbors.

(c) In this configuration all four tetrahedra share one edge.

Fig. 1. A polyhedron with a tetrahedral decomposition of four tetrahedra is in one of those three configurations. The shared vertex or edge is marked.

3.1 Preparation

Given the polyhedron P and a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra, we look at the dual graph $D(\Sigma)$ of the tetrahedral decomposition. For the rest of the section we want the dual graph to be a tree. This is possible by looking at a spanning tree T of $D(\Sigma)$ rooted at some arbitrary leaf node.

In the following, we will place beacons depending only on the neighborhood relation between tetrahedra. If T is missing some edge $\{u, v\}$ from $D(\Sigma)$ we "forget" that tetrahedra u and v are neighbors, i.e., share a common facet. We have less information about a tetrahedron's neighborhood and thus we might place more beacons than needed—but never less.

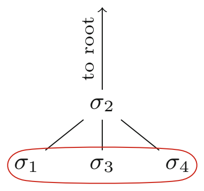
Note 3.2. In the following we will refer to nodes of T as well as their corresponding tetrahedra with σ_i. It should be clear from the context when the node and when the tetrahedron is meant—if not, it is indicated.

The main idea of the proof is as follows: In a recursive way we are going to place a beacon and remove tetrahedra until no tetrahedra are left. As will be shown, for every beacon we can remove at least three tetrahedra which yields the claimed upper bound. We will show this by induction and start with the base case:

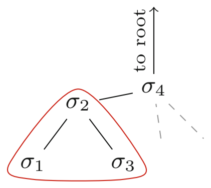
Lemma 3.3 (Base case). *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| \leq 4$ tetrahedra it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. If $m = 1$ then P is a tetrahedron and due to convexity no beacon is needed.

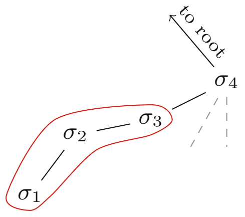
If $2 \leq m \leq 4$ we can apply Lemma 2.3 which shows that all tetrahedra share at least one common vertex v. We place the only beacon we are allowed to place at v. Then v is contained in every tetrahedron and thus, by convexity, every point in P can attract and be attracted by a beacon at v. $\square$



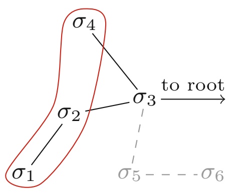
(a) Remove σ_1, σ_3, and σ_4 by placing a beacon where all four tetrahedra meet.



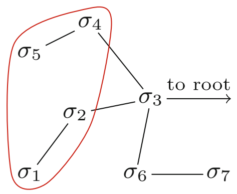
(b) Remove σ_1, σ_2, and σ_3 by placing a beacon where all four tetrahedra meet.



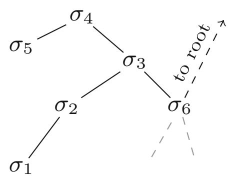
(c) Remove σ_1, σ_2, and σ_3 by placing a beacon where all four tetrahedra meet.



(d) Remove σ_1, σ_2, and σ_4 by placing a beacon where all four tetrahedra meet.



(e) Remove σ_1, σ_2, σ_4, and σ_5 by placing a beacon where σ_1 to σ_5 meet.



(f) The number and configuration of σ_6's children needs to be looked at.

Fig. 2. The possible configurations in the first part of the inductive step.

We can now proceed with the inductive step, that is, polyhedra with a tetrahedral decomposition of $m > 4$ tetrahedra. Our goal is to place k beacons which are contained in at least $3k + 1$ tetrahedra and can therefore mutually attract all points in those tetrahedra. Afterwards, we will remove at least $3k$ tetrahedra, leaving a polyhedron with a tetrahedral decomposition of strictly less than m tetrahedra, to which we can apply the induction hypothesis. We then need to show how to route between the smaller polyhedron and the removed tetrahedra.

To do this, we look at a deepest leaf σ_1 of the spanning tree T. If multiple leaves with the same depth exist we choose the one whose parent σ_2 has the largest number of children, breaking ties arbitrarily. In Fig. 2 we can see different cases how the part of T which contains σ_1 and σ_2 might look like. We first concentrate on Figs. 2a to e and show for them the first part of the inductive step. Note that in all five cases there needs to be at least one additional root node—either because we have strictly more than four tetrahedra or because the tree is required to be rooted at a leaf node. The second part of the inductive step, namely Fig. 2f will be dealt with in Lemma 3.6.

Lemma 3.4 (Inductive step I). *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let σ_1 be a deepest leaf of T with* *the maximum number of siblings and let σ_2 be its parent. Assume furthermore that any of the following conditions holds:*

(i) σ_2 has three children σ_1, σ_3, and σ_4 (see Fig. 2a),
(ii) σ_2 has two children σ_1 and σ_3 and a parent σ_4 (see Fig. 2b),
(iii) σ_2 has one child σ_1 and is the only child of its parent σ_3 whose parent is σ_4 (see Fig. 2c),
(iv) σ_2 has one child σ_1 and its parent σ_3 has two or three children of which one, σ_4, is a leaf (Fig. 2d), or
(v) σ_2 has one child σ_1 and its parent σ_3 has three children each of which has a single leaf child (Fig. 2e).

Then we can place one *beacon b at a vertex of σ_1 which is contained in at least four tetrahedra. We can then remove at least three tetrahedra containing b without violating the tree structure of T and while there is at least one tetrahedron left in T which contains b.*

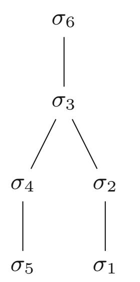
Proof. We show this individually for the conditions.

(i)–(iv) In all those cases the induced subgraph of the nodes σ_1, σ_2, σ_3, and σ_4 is connected. We can then see with Lemma 2.3(iii) that the four tetrahedra share at least one vertex at which b is placed.
After that we either remove σ_1, σ_3, and σ_4 (case (i)); σ_1, σ_2, and σ_3 (cases (ii) and (iii)); or σ_1, σ_2, and σ_4 (case (iv)). In all of those cases only leaves or inner nodes with all their children are removed which means that the tree structure of T is preserved. Additionally, we only remove three of the four tetrahedra that contain b, thus, one of them remains in T.

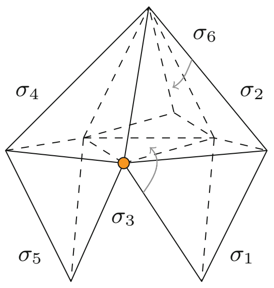
(v) Looking at Fig. 2e we see that we have three different sets, each containing σ_3, a child σ_i of σ_3, and σ_i's child: $\{\sigma_1, \sigma_2, \sigma_3\}$, $\{\sigma_5, \sigma_4, \sigma_3\}$, and $\{\sigma_7, \sigma_6, \sigma_3\}$.
When applying Lemma 2.3(ii), we see that each set shares one edge, giving us three edges of σ_3. Since at most two edges in any tetrahedron can be disjoint, at least two of the given edges must share a common vertex. Without loss of generality let these be the edges shared by $\{\sigma_1, \sigma_2, \sigma_3\}$ and $\{\sigma_5, \sigma_4, \sigma_3\}$. We can then place b at the shared vertex and afterwards remove σ_1, σ_2, σ_4, and σ_5. The beacon b is also contained in σ_3 which remains in T. □

3.2 Special Cases in the Inductive Step

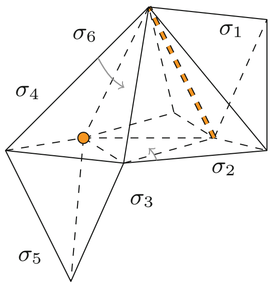
Until now, we have ignored the configuration in Fig. 2f. The problem here is that to remove the tetrahedra σ_1 to σ_5 we need to place two beacons. Placing two beacons but only removing five tetrahedra violates our assumption that we can always remove at least $3k$ tetrahedra by placing k beacons. If we removed σ_6 and σ_6 had additional children then T would no longer be connected which also leads to a non-provable situation. Thus, we need to look at the number and different configurations of the (additional) children of σ_6.



(a) The dual graph of the tetrahedral decomposition.



(b) All tetrahedra but the rearmost tetrahedron σ_6 share one common vertex, here marked in orange.



(c) The four tetrahedra on the left share a common vertex while the right four tetrahedra share a common edge.

Fig. 3. One tetrahedron σ_6 with a subtree of five tetrahedra. Subfigures (b) and (c) depict configurations that satisfy cases (i) and (ii) of Lemma 3.5, respectively. (Color figure online)

Since there are many different configurations of σ_6's children (and their subtrees) we decided to use a brute force approach to generate all cases we need to look at. Afterwards we removed all cases where Lemma 3.4 can be applied and all cases where only the order of the children differed. This leaves us with nine different cases where (obviously) the subtree from Fig. 2f is always present. Thus we seek more information from this specific configuration.

Lemma 3.5. *Given a tetrahedral decomposition of six tetrahedra with the dual graph as depicted in Fig. 3a. Then at least one of the following holds:*

(i) *σ_1 to σ_5 share a common vertex, or*
(ii) *σ_3, σ_4, σ_5, and σ_6 share a common vertex v; σ_1, σ_2, σ_3, and σ_6 share a common edge e; and $v \cap e = \emptyset$.*

Proof. We first define $S_1 = \{\sigma_3, \sigma_4, \sigma_5, \sigma_6\}$ and $S_2 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_6\}$. We observe that by Lemma 2.3 each set shares at least a vertex, but can also share an edge. We distinguish the cases by the shared geometric object:

- If both S_1 and S_2 each share an edge, case (i) holds. Each such edge needs to be part of the triangular facet which connects σ_3 and σ_6. Thus both edges share a common vertex.
- If just one of the two sets shares an edge e and the other shares only a vertex v there are two trivial cases: If $v \cap e = \emptyset$ then case (ii) is true—see Fig. 3c for an example. On the other hand, if $v \cap e = v$ then case (i) holds.
- The last case is the one in which each of the sets shares only a vertex. The situation can be seen in Fig. 3b. First look at the vertex v of σ_3 not contained in the facet shared by σ_3 and σ_6, i.e., $v \notin \sigma_3 \cap \sigma_6$. In the figure v is marked in orange. We observe that then all neighbors of σ_3 except σ_6 contain v. Thus, σ_2 contains v and three of its four facets are incident to v. One of the

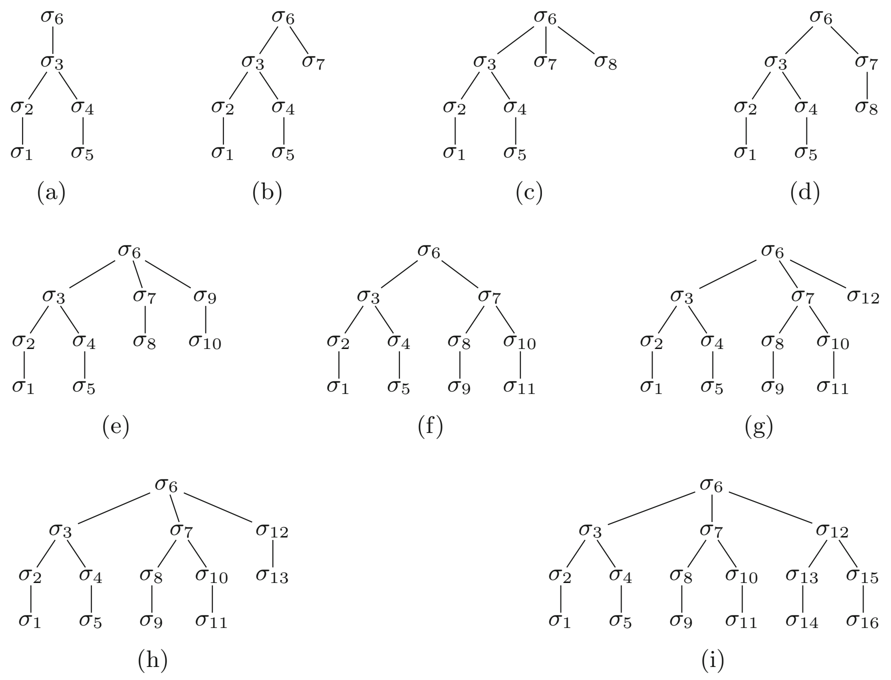


Fig. 4. All “nontrivial” configurations of children of σ_6. The tree in (a) is a subtree of all configurations. In all cases σ_6 has no other children than the ones shown here. Furthermore by the requirement that T is rooted at a leaf node, σ_6 needs to have an additional parent (except for case (a)).

facets is the shared facet with σ_3, but the other two are where σ_1 could be placed. σ_1 cannot be located at the fourth facet of σ_2, since it would then share an edge with σ_3 and σ_6 which is covered in the previous cases. Therefore, σ_1 contains v and with the same argument the same holds true for σ_5. Then case (i) holds. $\square$

Lemma 3.6 (Inductive step II). *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let $T' \subseteq T$ be a subtree of T with height 3 for which Lemma 3.4 cannot be applied. (See Fig. 4 for all possible cases.)*

In T' we can then place $k \geq 2$ beacons for which it holds that the beacons are contained in at least $3k + 1$ tetrahedra and the graph of the beacons and the edges they are contained in is connected.

We can then remove at least $3k$ tetrahedra from T', each of which contains a beacon, without violating the tree structure of T. After removal there is at least one tetrahedron left in T which contains one of the beacons.

Due to space constraints we omit the proof here. It involves analyzing each of the nine cases individually and carefully applying Lemma 3.5. The full proof can however be found in [8, Lemma 5.9, pp. 34–37].

Lemma 3.4 shows the inductive step for all subtrees with height at most 2 and Lemma 3.6 enumerates all combinatorically different subtrees with height exactly 3. Since in both lemmas the height of the subtree decreases by at least 1 we can always apply either of them.

3.3 Conclusion

We can now restate Hypothesis 3.1 as a theorem:

Theorem 3.7 (Upper bound). *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. We show this by induction. The base case is shown by Lemma 3.3. We assume that the induction hypothesis (Hypothesis 3.1) holds for all polyhedra with a tetrahedral decomposition with strictly less than m tetrahedra. We then show that it also holds for tetrahedral decompositions Σ with exactly m tetrahedra.

Look at a spanning tree T of the dual graph $D(\Sigma)$ of the tetrahedral decomposition Σ which is rooted at an arbitrary leaf node. Let σ_1 be a deepest leaf node and if σ_1 is not unique choose the one with the largest number of siblings, breaking ties arbitrarily. We can then apply either Lemma 3.4 or Lemma 3.6 and know at least the following:

(i) We have placed $k \geq 1$ beacons and removed at least $3k$ tetrahedra.
(ii) Every removed tetrahedron contains at least one beacon.
(iii) The induced subgraph of the placed beacons on the vertices and edges of the polyhedron is connected.
(iv) There is at least one beacon b contained in the remaining polyhedron P'.

From (i) it follows that the new polyhedron P' has a tetrahedral decomposition of $m' \leq m - 3k$ tetrahedra. We can then apply the induction hypothesis for P'. Thus we only need to place $k' = \lfloor \frac{m'+1}{3} \rfloor \leq \lfloor \frac{m-3k+1}{3} \rfloor = \lfloor \frac{m+1}{3} \rfloor - k$ beacons in P' to route between any pair of points in P'. Since $k' + k = \lfloor \frac{m+1}{3} \rfloor$ we never place more beacons than we are allowed.

From the induction hypothesis and (iv) we conclude that we are especially able to route from any point in P' to the beacon b and vice versa, since b is contained in P'. With (ii) we know that for every point p in the removed tetrahedra there is a beacon b' such that p attracts b' and b' attracts p. Finally, with (iii) we know that we can route between all beacons we have placed. This especially means that we can route from every beacon to the beacon b which is inside P' and vice versa.

This completes the inductive step and thus, by induction, we have proved the theorem. □

Observation 3.8. Placing $\max\left(1, \lfloor \frac{m+1}{3} \rfloor\right)$ beacons is always sufficient to cover a polyhedron with a tetrahedral decomposition with m tetrahedra. We need at least one beacon to cover a polyhedron and placing them as in the previous proof is enough.

4 A Lower Bound for Beacon-Based Routing

We now want to show a lower bound for the number of beacons needed to route within polyhedra with a tetrahedral decomposition. To do this we first show a different lower bound proof for two dimensions which can then be easily applied to three dimensions.

As shown by Biro et al. [6], $\lfloor \frac{n}{2} \rfloor - 1$ is not only an upper but also a lower bound for the necessary number of beacons in simple polygons. The idea for the following construction is similar to the one used by Shermer [12] for the lower bound for beacon-based routing in orthogonal polygons. We first show the construction of a class of spiral-shaped polygons for which we will then show that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are needed for a specific pair of points.

Definition 4.1. *For every $c \in \mathbb{N}^{\geq 1}$ and some small $0 < \delta < 1$ a c-*corner spiral polygon *is a simple polygon with $n = 2c + 2$ vertices. These vertices, given in counterclockwise order, are called s, q_1, q_2, $\ldots$, q_c, t, r_c, r_{c-1}, $\ldots$, r_1 and their coordinates are given in polar notation as follows:*

- $s = (1; 0\pi)$, $t = \left(\lfloor \frac{c+1}{3} \rfloor + 1; (c+1) \cdot \frac{2}{3}\pi\right)$,
- $q_k = \left(\lfloor \frac{k}{3} \rfloor + 1 + \delta; k \cdot \frac{2}{3}\pi\right)$ *for all* $1 \leq k \leq c$, *and*
- $r_k = \left(\lfloor \frac{k}{3} \rfloor + 1; k \cdot \frac{2}{3}\pi\right)$ *for all* $1 \leq k \leq c$.

The two vertices r_k and q_k form the k-th corner*. The trapezoids $\square r_k q_k q_{k+1} r_{k+1}$ for all $1 \leq k < c$ and the two triangles $\triangle s r_1 q_1$ and $\triangle t r_c q_c$ are each called a* hallway*.*

An example for $c = 5$ can be seen in Fig. 5. There are five corners and we have already placed five beacons to be able to route from s to t.

Lemma 4.2 (Two-dimensional lower bound). *Given a c-corner spiral polygon c beacons are necessary to route from s to t.*

Proof. To show that we need c beacons we introduce some additional notational conventions as depicted in Fig. 6a. The exterior angle at each (reflex) vertex r_k is called α_k. We split each hallway into two parts. To achieve this, three rays starting at the origin with the angles $\frac{1}{3}\pi$, π, and $\frac{5}{3}\pi$ (the dotted rays in Fig. 5) are drawn. Every hallway is divided by exactly one of the three rays and the intersection points with the polygon's boundary are called a_k and b_k: a_k is the intersection of one of the rays with the edge $r_{k-1}r_k$ and b_k with the edge $q_{k-1}q_k$.

We observe that, due to the triangular shape, the angle α_k is always strictly less than $90°$, for every $1 \leq k \leq c$.

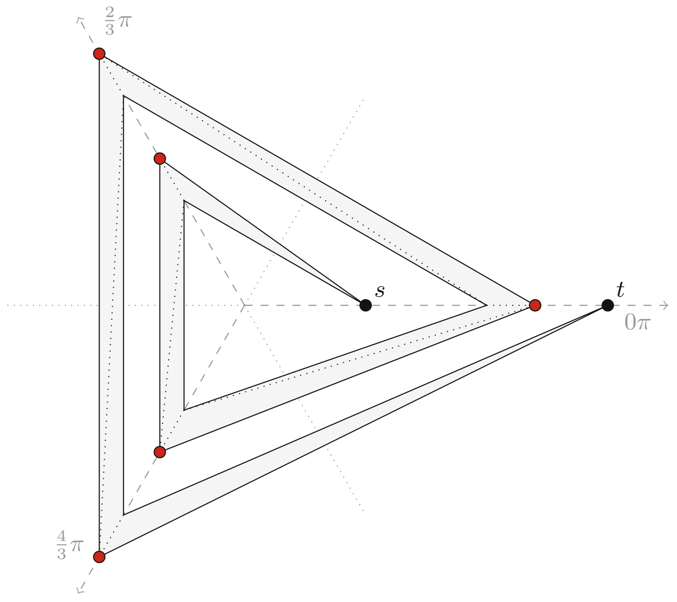


Fig. 5. A 5-corner spiral polygon with $\delta = 0.4$ for which five beacons (marked in red) are necessary to route from s to t. (Color figure online)

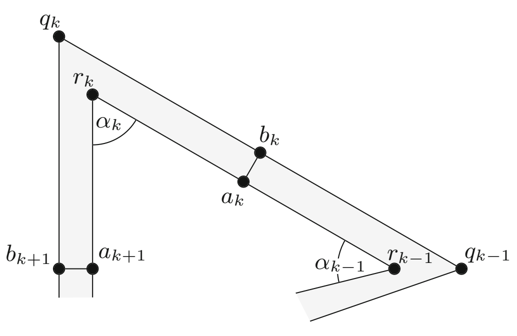
We now divide our polygon into $c + 2$ subpolygons C_0 to C_{c+1} at each pair a_k and b_k. This subdivision results in two triangles C_0 and C_{c+1} which contain s and t, respectively, and c subpolygons C_1 to C_c which are called *complete corners* and which all have the same structure. We show that every such complete corner needs to contain at least one beacon to be able to route from s to t. We look at one complete corner C_k, $1 \leq k \leq c$, shown in Fig. 6b.

We want to route from C_{k-1} to C_{k+1}. To route any point from C_{k-1} (which all lie to the right of $a_k b_k$) towards b_{k+1} there has to be a beacon such that the shortest line segment of this beacon to the line segment $r_k a_k$ ends in r_k. Otherwise, an attracted point will get stuck on $r_k a_k$ because the shortest path ends somewhere on $r_k a_k$ (excluding r_k itself). In Fig. 6b we see the case where b_{k+1} is a beacon. Here d is a dead point with respect to b_{k+1} and no point from C_{k-1} will travel further into C_k when attracted by b_{k+1}.

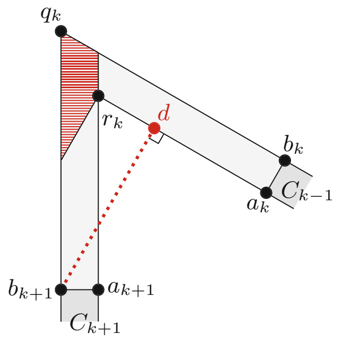
The marked region in Fig. 6b is the region in which every point can attract at least all points on the line segment $a_k b_k$. Additionally, every point in the region can be attracted by a beacon somewhere in the region of C_{k+1} to the left of the line trough $a_{k+1} r_k$, i.e., the part of C_{k+1} before turning at r_{k+1}.

On the other hand, there is no better option than b_{k+1} for a beacon position outside of C_k. All other points in C_{k+1} lie to the right of the line through $b_{k+1} d$ and hence their respective dead point on $r_k r_{k-1}$ lies further away from r_k.

We can see that if the length of the hallways (the distance between r_k and r_{k+1}) is sufficiently large compared to their width (the distance between a_k and b_k) it is



(a) Notation for various parts on the triangular spiral.



(b) The complete corner C_k in lighter gray.

Fig. 6. A more detailed look at the parts of the spiral polygon.

never possible for a point outside of C_k to be inside the marked region. Therefore it is not possible to route from somewhere inside C_{k-1} to somewhere inside C_{k+1} without an additional beacon inside C_k. $\square$

We now apply the idea of this proof to three dimensions starting with the definition of the spiral polyhedron.

Definition 4.3. *For every $c \in \mathbb{N}^{\geq 1}$ and some small $0 < \delta < 1$ a c-*corner spiral polyhedron *is a polyhedron with $n = 3c + 2$ vertices. These vertices are s and t as well as q_k, r_k, and z_k for all $1 \leq k \leq c$. The coordinates of s, t, q_k, and r_k are the same as in Definition 4.1 with their z-coordinate set to 0. The z_k are positioned above r_k or more formally $z_k := r_k + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ for all $1 \leq k \leq c$.*

The edges and facets of the polyhedron are given by the tetrahedral decomposition:

- *The start and end tetrahedra are formed by $\triangle r_1 q_1 z_1 s$ and $\triangle r_c q_c z_c t$.*
- *The* hallway *between two triangles $\triangle r_k q_k z_k$ and $\triangle r_{k+1} q_{k+1} z_{k+1}$ consists of the three tetrahedra $\triangle r_k q_k z_k r_{k+1}$, $\triangle r_{k+1} q_{k+1} z_{k+1} q_k$, and $\triangle q_k z_k r_{k+1} z_{k+1}$.*

The three vertices r_k, q_k, and z_k form the k-th corner*.*

Observation 4.4. The smallest c-corner spiral polyhedron with $c = 1$ consists of exactly two tetrahedra. For greater c we add exactly $c - 1$ hallways, each consisting of three tetrahedra. This means that a c-corner spiral polyhedron has a tetrahedral decomposition with $m = 3c - 1$ tetrahedra. It follows from Definition 4.3 that the number of tetrahedra relative to the number of vertices is $m = 3 \cdot \frac{n-2}{3} - 1 = n - 3$.

Lemma 4.5 (Lower bound). *Given a c-corner spiral polyhedron P, c beacons are necessary to route from s to t.*

Proof. We project P onto the xy-plane which results in a c-corner spiral polygon P' due to the construction of the c-corner spiral polyhedron. To P' we can apply Lemma 4.2 where we showed that c beacons (placed in an area around each of the c corners) are sometimes necessary to route in P'.

As opposed to the polygon, the movement in the polyhedron is not constrained to the xy-plane. Additionally, beacons can be placed at locations which do not lie in the xy-plane. We need to show that this does not change the situation in a way so that less than c beacons are necessary.

First, note that, due to the construction in Definition 4.3, every cross section of the polyhedron parallel to the xy-plane yields a c-corner spiral polygon with different widths δ. For every such cross section, Lemma 4.2 tells us that to route only in this cross section, c beacons are needed.

Additionally, the hallway's inner boundary $r_k z_k r_{k+1} z_{k+1}$ is perpendicular to the xy-plane. This means that the movement of all points p which are attracted by a beacon b can be split into a xy-movement and a z-movement because the z-coordinate is not important for any movement along the inner boundary. Since each hallway is convex there is no other movement of a point p attracted by a beacon b which is constrained by the polyhedron's boundary ∂P. We can then only look at the xy-movement which again yields a two-dimensional situation to which Lemma 4.2 can be applied. □

5 A Sharp Bound for Beacon-Based Routing

Theorem 5.1. *Given a polyhedron P for which a tetrahedral decomposition with m tetrahedra exists, it is always sufficient and sometimes necessary to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. In Theorem 3.7 we have shown that $\lfloor \frac{m+1}{3} \rfloor$ is an upper bound.

For any given m we can construct a c-corner spiral polyhedron P_m with $c = \lfloor \frac{m+1}{3} \rfloor$ corners for which, by Lemma 4.5, c beacons are necessary. The number of tetrahedra in P_m is $m' = 3c - 1$ (see Observation 4.4) and this is also the smallest number of tetrahedra in any tetrahedral decomposition of P_m: If there was a tetrahedral decomposition with less tetrahedra then by Theorem 3.7 less than c beacons would be needed which contradicts Lemma 4.5.

If $m' < m$, i.e. due to the flooring function the c-corner spiral contains one or two tetrahedra less than m, we add the missing tetrahedra as if constructing a $c + 1$-corner spiral. This does not lead to less beacons being needed. □

6 Conclusion

We have shown that the problem of finding a minimal beacon set in a polyhedron P to route between all pairs of points or all points and a specific point is NP-hard and APX-hard. This holds also true for the problem of finding a minimal beacon set to cover a polyhedron P.

We have shown that, given a tetrahedral decomposition of a polyhedron P with m tetrahedra it is always sufficient to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P. We then gave a class of polyhedra for which this upper bound is always necessary.

A lot of questions which have been answered by various authors in two dimensions remain open for the three-dimensional case. They include learning about the complexity of finding an optimal beacon set to route between a given pair of points. Additional open questions are about attraction regions (computing the set of all points attracted by a single beacon) and beacons kernels (all points at which a beacon can attract all points in the polyhedron).

Furthermore Cleve [8] has shown that not all polyhedra can be covered by beacons placed at the polyhedron's vertices and Aldana-Galván et al. [1,2] showed that this is even true for orthogonal polyhedra. Given a tetrahedral decomposition of a polyhedron it remains open whether it is possible to cover a polyhedron with less than $\max\left(1, \lfloor \frac{m+1}{3} \rfloor\right)$ beacons as seen in Observation 3.8. It seems challenging to further look at the beacon-coverage problem in general polyhedra.

Acknowledgments. We thank the anonymous reviewers for their thorough reading of the paper and helpful suggestions.

References

1. Aldana-Galván, I., Álvarez-Rebollar, J.L., Catana-Salazar, J.C., Marín-Nevárez, N., Solís-Villarreal, E., Urrutia, J., Velarde, C.: Beacon coverage in orthogonal polyhedra. In: 29th Canadian Conference on Computational Geometry (CCCG 2017), Ottawa, pp. 166–171, July 2017
2. Aldana-Galván, I., Álvarez-Rebollar, J.L., Catana-Salazar, J.C., Marín-Nevárez, N., Solís-Villarreal, E., Urrutia, J., Velarde, C.: Covering orthotrees with guards and beacons. In: XVII Spanish Meeting on Computational Geometry (XVII ECG), Alicante, June 2017
3. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. Comput. Euclidean Geom. **4**, 47–123 (1995)
4. Biro, M.: Beacon-based routing and guarding. Ph.D. thesis, State University of New York at Stony Brook (2013)
5. Biro, M., Gao, J., Iwerks, J., Kostitsyna, I., Mitchell, J.S.B.: Beacon-based routing and coverage. In: 21st Fall Workshop on Computational Geometry (FWCG 2011) (2011)
6. Biro, M., Gao, J., Iwerks, J., Kostitsyna, I., Mitchell, J.S.B.: Combinatorics of beacon-based routing and coverage. In: Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG 2013), vol. 1, p. 3 (2013)
7. Chazelle, B.: Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. SIAM J. Comput. **13**(3), 488–507 (1984)
8. Cleve, J.: Combinatorics of beacon-based routing and guarding in three dimensions. Master's thesis, Freie Universität Berlin, Berlin, March 2017
9. Ghosh, S.K.: Visibility Algorithms in the Plane. Cambridge University Press, Cambridge (2007)

10. Lennes, N.J.: Theorems on the simple finite polygon and polyhedron. Am. J. Math. **33**(1/4), 37 (1911)
11. Ruppert, J., Seidel, R.: On the difficulty of triangulating three-dimensional non-convex polyhedra. Discret. Comput. Geom. **7**(3), 227–253 (1992)
12. Shermer, T.C.: A combinatorial bound for beacon-based routing in orthogonal polygons. arXiv preprint arXiv:1507.03509 (2015)

On Split B_1-EPG Graphs

Zakir Deniz[1](✉), Simon Nivelle[2], Bernard Ries[3], and David Schindl[4]

[1] Duzce University, Duzce, Turkey
zakirdeniz@duzce.edu.tr
[2] ENS Paris Saclay, Paris, France
snivelle@ens-paris-saclay.fr
[3] University of Fribourg, Fribourg, Switzerland
bernard.ries@unifr.ch
[4] Geneva School of Business Administration, Geneva, Switzerland
david.schindl@hesge.ch

Abstract. In this paper, we are interested in edge intersection graphs of paths in a grid, such that each such path has at most one bend. These graphs were introduced in [12] and they are called B_1-EPG graphs. In particular, we focus on split graphs and characterise those that are B_1-EPG. This characterisation allows us to disprove a conjecture of Cameron et al. [7]. The existence of polynomial-time recognition algorithm for this graph class is still unknown. We furthermore investigate inclusion relationships among subclasses of split graphs that are B_1-EPG.

1 Introduction

Golumbic et al. introduced in [12] the notion of *edge intersection graphs of paths in a grid* (referred to as *EPG graphs*). An undirected graph $G = (V, E)$ is called an *EPG graph*, if one can associate a path in a rectangular grid with each vertex such that two vertices are adjacent if and only if the corresponding paths intersect on at least one grid-edge. The authors showed in [12] that every graph is in fact an EPG graph. Therefore, they introduced additional restrictions on the paths by limiting the number of *bends* (a bend is a 90° turn of a path at a grid-point) that a path can have. An undirected graph $G = (V, E)$ is then called a *B_k-EPG graph*, for some integer $k \geq 0$, if one can associate with each vertex a path with at most k bends in a rectangular grid such that two vertices are adjacent if and only if the corresponding paths intersect on at least one grid-edge.

One motivation for introducing these graphs comes from chip manufacturing. Indeed, each wire on a chip can be seen as a path on a rectangular grid. Since each wire bend requires a so-called transition hole, and since a large number of such holes increase the layout area as well as the overall cost of the chip, it is of interest to limit the total number of holes respectively to limit the number of bends per wire. Another motivation comes from the fact that B_k-EPG graphs generalize the well-known class of interval graphs. From its definition, it is easy to see that the class of B_0-EPG graphs is indeed equivalent to the class of interval graphs.

https://doi.org/10.1007/978-3-319-77404-6_27

Since the introduction of the notion of B_k-EPG graphs, there has been a lot of research done on these graphs from several points of view (see for instance [1–8,10,11,13–15]). Since B_0-EPG graphs coincide with the class of interval graphs, particular attention has been paid to the class of B_1-EPG graphs. The authors in [13] showed that recognizing B_1-EPG graphs is an NP-complete problem; the same holds for B_2-EPG graphs as recently shown in [15]. In any representation of a B_1-EPG graph, each path can only have one of the following four shapes: ⌞, ⌝, ⌜, ⌟ (a path with only a horizontal part or only a vertical part can be considered as a degenerate path of one of the four shapes mentioned before). In [7], the authors analysed B_1-EPG graphs for which the number of different shapes is restricted to a subset of the set above. They showed that testing membership to each of these restricted classes is also NP-complete. Furthermore, they focused on chordal graphs that are B_1-EPG with the additional restriction that only one particular shape (namely ⌞) is allowed for all paths. In particular, they state a conjecture concerning the characterisation of split graphs that are B_1-EPG and where only paths with an ⌞ shape are allowed, by a family of forbidden induced subgraphs. Indeed, they present a list of nine forbidden induced subgraphs and conjecture that these are the only ones. In this paper, we disprove this conjecture by providing an additional forbidden induced subgraph. However, giving a complete list of forbidden induced subgraphs or deciding whether such a finite list exists remains open. Furthermore, we provide a characterisation of split graphs that are B_1-EPG and where only paths with an ⌞ shape are allowed. Notice that this characterisation does not imply a polynomial-time recognition algorithm. In addition, for any subset P of the four possible shapes mentioned above, we investigate inclusion relationships among subclasses of split graphs that are B_1-EPG and where only shapes from P are allowed.

Our paper is organised as follows. In Sect. 2, we present definitions and notations as well as some preliminary results and observations that we will use throughout the paper. Section 3 deals with the inclusion relationships among subclasses of split graphs that are B_1-EPG. In Sect. 4, we present a characterisation of split B_1-EPG graphs and disprove the conjecture of Cameron et al. [7]. We finish with Sect. 5 in which we also mention further results that we obtained and suggest some further research directions.

2 Preliminaries

We only consider finite, undirected graphs that have no self-loops and no multiple edges. We refer to [9] or [16] for undefined terminology. Let $G = (V, E)$ be a graph. For a subset $S \subseteq V$, we let $G[S]$ denote the subgraph of G *induced* by S, which has vertex set S and edge set $\{uv \in E \mid u, v \in S\}$. We write $H \subseteq_i G$ if a graph H is an induced subgraph of G. Moreover, for a vertex $v \in V$, we write $G - v = G[V \backslash \{v\}]$ and for a subset $V' \subseteq V$, we write $G - V' = G[V \backslash V']$. The set of vertices adjacent to some vertex u is called the *neighborhood of* u and will be denoted by $N(u)$. The *closed neighborhood of* u is defined as $N[u] = N(u) \cup \{u\}$. A vertex u *dominates* some adjacent (resp. non-adjacent) vertex v if $N[v] \subseteq N[u]$

(resp. if $N(v) \subseteq N(u)$). Two vertices u, v in G are said to be *comparable* if u dominates v or v dominates u. Two vertices that are not comparable are said to be *incomparable.* A *split graph* is a graph $G = (V, E)$ whose vertex set V can be partitioned into a clique K (i.e., a set of pairwise adjacent vertices) and a stable set S (i.e., a set of pairwise non-adjacent vertices). We say that (K, S) is a *split partition of* G. The vertices in S will be called the *S-vertices.*

Let $\mathcal{G}$ be a rectangular grid of size $m \times m'$. The horizontal grid lines will be referred to as *rows* and denoted by $x_0, x_1, \cdots, x_{m-1}$ and the vertical grid lines will be referred to as *columns* and denoted by $y_0, y_1, \cdots, y_{m'-1}$. As already mentioned above, in any representation of a B_1-EPG graph, each path can only have one of the following four possible shapes: ⌞, ⌝, ⌜, ⌟. A path with an ⌞-shape will be called an *⌞-path.* In a similar way we define ⌝-path, ⌜-path and ⌟-path. For any subset P of the four possible shapes, we denote by $[P]$ the class of B_1-EPG graphs which admit a representation in which each path has one of the shapes in P. In particular, we denote by $[P]_s$ the class of B_1-EPG split graphs which admit a representation in which each path has one of the shapes in P. For simplicity, if P contains all four shapes, we write B_1-EPG$_s$. A representation of a B_1-EPG graph containing only paths with a shape in P is called a $[P]$-representation.

Let $G = (V, E)$ be a B_1-EPG graph and let $v \in V$. We denote by P_v the path representing v in a B_1-EPG representation of G. Consider a clique K (resp. a stable set S) in G. Any path representing a vertex in K (resp. in S) will simply be referred to as a *path of* K (resp. *path of* S). Concerning cliques, the following useful lemma has been shown in [12].

Lemma 1. *Let $G = (V, E)$ be a B_1-EPG graph. In any B_1-EPG representation of G, a clique K of G is represented either as an edge-clique or as a claw-clique (see Fig. 1).*

Notice that in an edge-clique, all paths share a common grid-edge, called the *base of the clique*, while in a claw-clique, all paths share a common grid-point, called the *center of the clique.*

A *gem* is a graph with vertex set $\{c_1, c_2, c_3, s_1, s_2\}$ and edge set $\{s_1c_1, s_1c_2, c_1c_2, c_2c_3, c_1c_3, s_2c_2, s_2c_3\}$ (see Fig. 2(a)). It is easy to see that a gem, as an induced subgraph of a split graph $G = (V, E)$ with split partition (K, S), must satisfy $c_1, c_2, c_3 \in K$ and $s_1, s_2 \in S$. A *bull* is a graph with vertex set $\{c_1, c_2, s_1, s_2, s_3\}$ and edge set $\{c_1c_2, c_1s_2, c_2s_2, c_1s_1, c_2s_3\}$ (see Fig. 2(b)). Again, it is easy to see that a bull, as an induced subgraph of a split graph $G = (V, E)$ with split partition (K, S), must satisfy $c_1, c_2 \in K$ and $s_1, s_3 \in S$. In the case

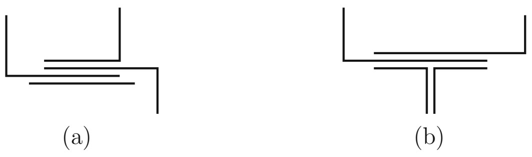


Fig. 1. An edge-clique (a) and a claw-clique (b).

where $s_2 \in S$ as well, the bull is called an S-*bull*. Gems and S-bulls have played an important role in [7]. As we will see, they are also crucial in our results.

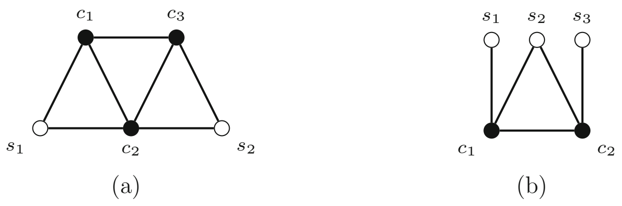


Fig. 2. (a) A gem. (b) An S-bull.

The following definitions have been introduced in [7]. Let $G = (V, E)$ be in $[\llcorner]_s$ with split partition (K, S). Consider an $[\llcorner]_s$-representation of G. Clearly, the clique K must be represented as an edge-clique. This grid-edge is called the *base*. Without loss of generality, we may assume that the base is vertical. The horizontal parts of the paths representing vertices in K are called *branches*. Let F be the vertical line-segment which is the union of the vertical parts of all paths representing vertices in K. The part of F below the base is called the *trunk*. The part of F above trunk is called the *crown* (see Fig. 3).

The following three observations have been made in [7]. As we will see, they will be very helpful in the proof of our main results.

Observation 1 *([7]). Let $G = (V, E)$ be a split graph in $[\llcorner]_s$. Then, the S-vertices whose paths lie on the same branch (or on the crown) are pairwise comparable. Furthermore, an S-vertex whose path lies on the trunk dominates all S-vertices whose paths lie below it in the representation.*

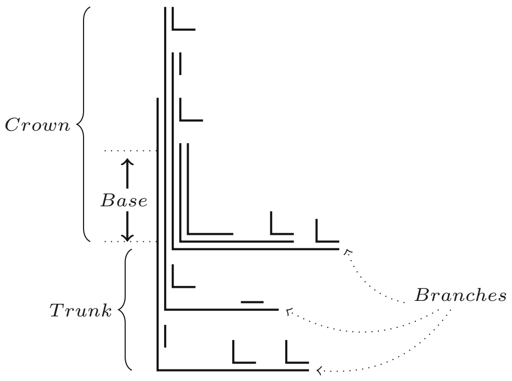


Fig. 3. An $[\llcorner]$-representation of a split graph with the notions of crown, base, branches and trunk.

Observation 2 *([7]). Let $G = (V, E)$ be a split graph in $[\llcorner]_s$. If G contains a gem, then exactly one of the gem's S-vertices has its path lying on the crown of the representation.*

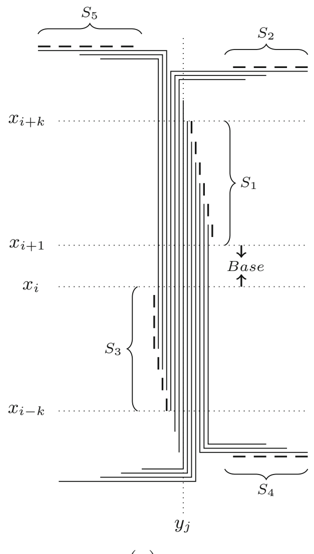
Observation 3 *([7]). Let $G = (V, E)$ be a split graph in $[\llcorner]_s$. If G contains an S-bull, then some S-vertices of this bull have their paths lying on either the crown or trunk of the representation.*

It is easy to see that we can generalize Observation 1 in the following way.

Observation 4. *Let $G = (V, E)$ be a split graph in B_1-EPG$_s$ with split partition (K, S).*

Assume that K is represented as an edge-clique with base going from (x_i, y_j) to (x_{i+1}, y_j) (see Fig. 4(a)). Then, the S-vertices whose paths use column y_j above (x_{i+1}, y_j), say between rows x_{i+1} and x_{i+k} (resp. below (x_i, y_j), say between rows x_{i-k} and x_i) and the S-vertices whose paths use some row $x_{i+\ell}$, $\ell \geq k$ (resp. $x_{i-\ell}$, $\ell \geq k$) on a same side of y_j (right or left) are pairwise comparable.

Similarly, assume that K is represented as a claw-clique with center (x_i, y_j) and assume that no path of K uses the grid-edge going from (x_i, y_{j-1}) to (x_i, y_j) (see Fig. 4(b)). Then, the S-vertices whose paths use column y_j above (x_i, y_j), say between rows x_{i+1} and x_{i+k} (resp. below (x_i, y_j), say between rows x_{i-k} and x_{i-1}) and the S-vertices whose paths use some row $x_{i+\ell}$, $\ell \geq k$ (resp. $x_{i-\ell}$, $\ell \geq k$) on a same side of y_j are pairwise comparable. Furthermore, the S-vertices whose paths use row x_i to the right of (x_i, y_j) are also pairwise comparable.



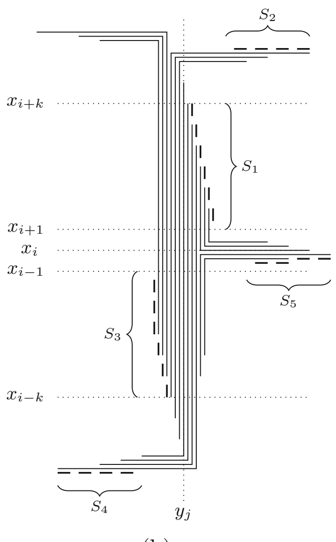


Fig. 4. (a) Vertices in $S_1 \cup S_2$ (resp. $S_1 \cup S_5$, $S_3 \cup S_4$) are pairwise comparable. (b) Vertices in $S_1 \cup S_2$ (resp. $S_3 \cup S_4$) are pairwise comparable; also the vertices of S_5 are pairwise comparable.

3 Subclasses of B_1-EPG_s

In [7], the authors showed that $[\llcorner] \subsetneq [\llcorner, \urcorner], [\llcorner, \ulcorner] \subsetneq [\llcorner, \ulcorner, \urcorner] \subsetneq B_1$-$EPG$ and that the two classes $[\llcorner, \urcorner], [\llcorner, \ulcorner]$ are incomparable. Here, we obtain a similar result when restricted to split graphs.

Theorem 1. $[\llcorner]_s \subsetneq [\llcorner, \urcorner]_s \subsetneq [\llcorner, \ulcorner]_s \subsetneq [\llcorner, \ulcorner, \urcorner]_s \subsetneq B_1\text{-}EPG_s$.

Notice that for split graphs we have $[\llcorner, \urcorner]_s \subsetneq [\llcorner, \ulcorner]_s$. We will prove Theorem 1 by a series of four lemmas (Lemmas 2, 3, 4 and 5). We first start with a useful proposition.

Proposition 1. *Consider a B_1-EPG representation of a gem (see Fig. 2(a)). Let $K = \{c_1, c_2, c_3\}$ and $S = \{s_1, s_2\}$. If K is represented as an edge-clique with base going from (x_i, y_j) to (x_{i+1}, y_j) or if K is represented as a claw-clique with center (x_i, y_j) and no path of K uses the grid-edge going from (x_i, y_{j-1}) to (x_i, y_j), then at least one of P_{s_1}, P_{s_2} intersects paths of K on column y_j.*

Proof. Consider a B_1-EPG representation of a gem with $K = \{c_1, c_2, c_3\}$ and $S = \{s_1, s_2\}$. Suppose that K is represented as an edge-clique with base going from (x_i, y_j) to (x_{i+1}, y_j) or K is represented as a claw-clique with center (x_i, y_j) and no path of K uses the grid-edge going from (x_i, y_{j-1}) to (x_i, y_j). By contradiction assume that both P_{s_1}, P_{s_2} do not intersect paths of K on column y_j. Since all paths have at most one bend, it follows that both P_{s_1}, P_{s_2} intersect paths of K on rows. Since s_1, s_2 have a common neighbour, P_{s_1}, P_{s_2} must intersect paths of K on a same row x_k either both to the right of y_j or both to the left of y_j. But this is not possible since s_1, s_2 are incomparable (see Observation 4).

Lemma 2. $[\llcorner]_s \subsetneq [\llcorner, \urcorner]_s$.

Proof. We clearly have $[\llcorner]_s \subseteq [\llcorner, \urcorner]_s$. Consider the graph G_4 in Fig. 5(a). We know from [7] that G_4 is not in $[\llcorner]_s$. But it is easy to see that G_4 is in $[\llcorner, \urcorner]_s$ (see Fig. 5(b)). Thus, $[\llcorner]_s \subsetneq [\llcorner, \urcorner]_s$.

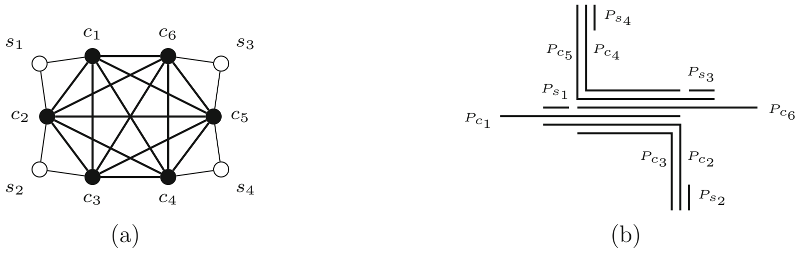


Fig. 5. The graph G_4 and a $[\llcorner, \urcorner]$-EPG representation of it.

Since G_4 is a minimal forbidden induced subgraph for the class $[\llcorner]_s$, it is minimal under inclusion with the property that it belongs to $[\llcorner, \urcorner]_s \setminus [\llcorner]_s$. However, there may exist other examples with fewer vertices.

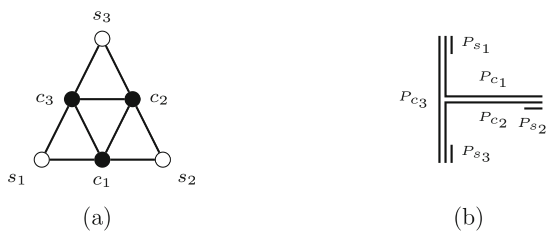


Fig. 6. The graph G_1 and a $[\llcorner, \ulcorner]$-EPG representation of it.

Lemma 3. $[\llcorner, \urcorner]_s \subsetneq [\llcorner, \ulcorner]_s$.

Proof. Consider the graph G_1, also called the 3-*sun*, in Fig. 6(a). It is clearly a split graph, and it has been shown in [7] to belong to $[\llcorner, \ulcorner] \backslash [\llcorner, \urcorner]$ (see Fig. 6(b) for a $[\llcorner, \ulcorner]$-EPG representation of it).

So we conclude that $[\llcorner, \urcorner]_s \neq [\llcorner, \ulcorner]_s$. It remains to show that $[\llcorner, \urcorner]_s \subset [\llcorner, \ulcorner]_s$. Consider a split graph G in $[\llcorner, \urcorner]_s$ with split partition (K, S). It follows from Lemma 1, that K must be represented as an edge-clique. Without loss of generality, we may assume that the base of K is vertical and goes from (x_i, y_j) to (x_{i+1}, y_j). Notice that, since only $\llcorner$-paths and $\urcorner$-paths are allowed, we may assume that each path of S intersects paths of K either with its vertical part or with its horizontal part, but never with both and thus, it is a degenerate path. Notice that no $\urcorner$-path has its horizontal part below (x_{i+1}, y_j), and no $\llcorner$-path has its horizontal part above (x_i, y_j). We may therefore transform the part above (x_{i+1}, y_j) of the whole representation by a symmetry with respect to column y_j, resulting in a $[\llcorner, \ulcorner]$ representation of G. Thus, $[\llcorner, \urcorner]_s \subsetneq [\llcorner, \ulcorner]_s$.

Notice that the symmetry used in the proof of Lemma 3 could not be used if one wanted to show that $[\llcorner, \ulcorner]_s \subsetneq [\llcorner, \urcorner]_s$, since the graph may have its clique represented by a claw-clique. Therefore, $[\llcorner, \ulcorner]_s \backslash [\llcorner, \urcorner]_s$ is exactly the set of those $[\llcorner, \ulcorner]_s$-EPG graphs admitting no split partition (K, S) such that K can be represented by an edge-clique. Notice also that since G_1 is the smallest graph not in $[\llcorner]_s$ (see [7]), it is also the smallest graph in $[\llcorner, \ulcorner]_s \backslash [\llcorner, \urcorner]_s$.

Lemma 4. $[\llcorner, \ulcorner]_s \subsetneq [\llcorner, \ulcorner, \urcorner]_s$.

Proof. We clearly have $[\llcorner, \ulcorner]_s \subseteq [\llcorner, \ulcorner, \urcorner]_s$. Let us consider the graph G_5 which belongs to $[\llcorner, \ulcorner, \urcorner]_s$ (see Fig. 7(a) and (b)). We will show that G_5 does not belong to $[\llcorner, \ulcorner]_s$.

By contradiction, assume that G_5 belongs to $[\llcorner, \ulcorner]_s$. We will distinguish two cases. First, suppose that the clique K induced by $\{c_1, \cdots, c_8\}$ is represented as an edge-clique. Without loss of generality, we may assume that the base of K is vertical and goes from (x_i, y_j) to (x_{i+1}, y_j). Since the vertex set $\{c_1, c_2, c_3, s_1, s_2\}$ induces a gem, it follows from Proposition 1 that at least one of P_{s_1}, P_{s_2} intersects paths of K on column y_j, say P_{s_1}. Also, we may assume, without loss

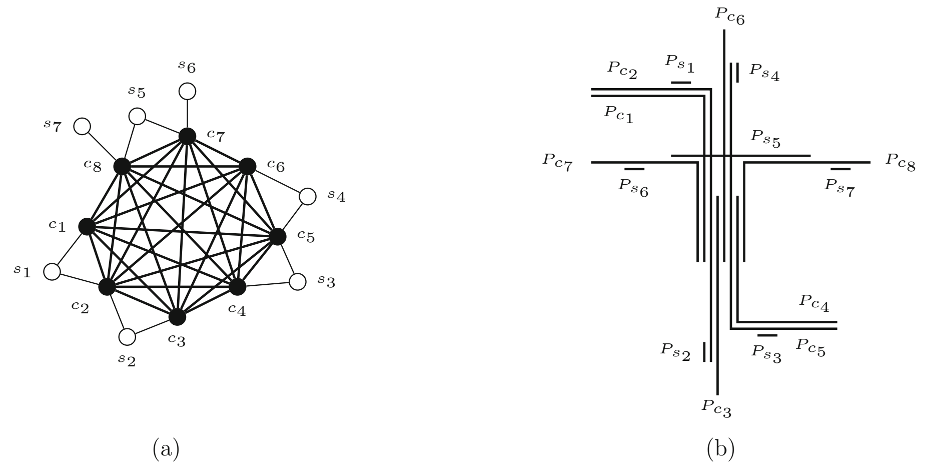


Fig. 7. The graph G_5 and a [$\llcorner$, $\ulcorner$, $\urcorner$]-EPG representation of it.

of generality, that it intersects paths of K above row x_{i+1}, say above row x_{r_1}, $r_1 \geq i+1$. Using the same argument for the gem induced by $\{c_4, c_5, c_6, s_3, s_4\}$, we may assume that P_{s_3} intersects path of K on column y_j. Since s_1, s_3 are incomparable, it follows that P_{s_3} lies below row x_i, say below row x_{r_3}, $r_3 \leq i$ (see Observation 4). Now consider the S-bull induced by $\{c_7, c_8, s_5, s_6, s_7\}$. Since c_7, c_8 are non-adjacent to s_1, s_3, their paths do neither go above row x_{r_1}, nor below row x_{r_3}. Since s_5, s_6, s_7 are non-adjacent to $c_1, \cdots, c_6$, it follows that $P_{s_5}, P_{s_6}, P_{s_7}$ cannot intersect paths of K on column y_j. Therefore, they must intersect paths of K on some same row x_r (since s_5 is adjacent to both c_7, c_8) and these intersections are all to the right of y_j (since we only allow $\llcorner$-paths and $\ulcorner$-paths. But this is clearly impossible, since s_6 and s_7 are incomparable. So we conclude that K cannot be represented as an edge-clique.

So we may assume now that K is represented as a claw-clique with center (x_i, y_j). Without loss of generality, we may assume that only paths of K use the grid edges going from (x_{i-1}, y_j) to (x_{i+1}, y_j). Clearly, all paths of S intersecting paths of K on row x_i must be pairwise comparable (see Observation 4). Hence, we immediately see that there are at most two such paths of S.

First assume there are exactly two paths of S intersecting paths of K on row x_i. Then, these must be paths P_{s_5}, P_{s_6} (resp. P_{s_5}, P_{s_7}). Without loss of generality, we may assume that c_7 is represented as a $\ulcorner$-path and c_8 is represented as an $\llcorner$-path. Notice that every other path of K with bend point (x_i, y_j) can be transformed into a vertical path (by deleting its horizontal part and extending it to (x_{i-1}, y_j) if it is an $\llcorner$-path and to (x_{i+1}, y_j) if it is a $\ulcorner$-path), since it does not intersect any path on row x_i. Hence, P_{c_7}, P_{c_8} are the only paths of K with bend point (x_i, y_j). Now, since c_7 (resp. c_8) has only two neighbours in S, namely s_5, s_6 (resp. s_5, s_7), it follows that it does not intersect any path of S with its vertical part. So we may transform P_{c_7} (resp. P_{c_8}) into an $\llcorner$-path (resp. a $\ulcorner$-path) with vertical part going from (x_i, y_j) to (x_{i+1}, y_j) (resp. to (x_{i-1}, y_j)). Hence, K is representable as an edge-clique. But we know from the above that this is not possible.

So let us now assume, that exactly one path of S is intersecting paths of K on row x_i. As before, notice that every path of K with bend point (x_i, y_j) not intersecting any path of S on row x_i can be transformed into a vertical path (by deleting its horizontal part and extending it to (x_{i-1}, y_j) if it is an $\llcorner$-path and to (x_{i+1}, y_j) if it is a $\ulcorner$-path). So this unique path of S intersecting paths of K on row x_i represents a vertex of degree at least two, hence one of $s_1, \cdots, s_5$ (otherwise we obtain again the case where K is represented as an edge-clique). First assume it represents s_1 (the cases when it represents s_2, s_3 or s_4 can be handled similarly). In other words, P_{c_1}, P_{c_2} are the only paths of K using row x_i. We may assume, without loss of generality, that c_1 is represented by a $\ulcorner$-path and c_2 by an $\llcorner$-path. Since c_1 does not have any neighbour in S except s_1, it follows that it does not intersect any path of S with its vertical part. Thus, as in the previous case, we can transform P_{c_1} into an $\llcorner$-path with vertical part going from (x_i, y_j) to (x_{i+1}, y_j). But then K is again represented as an edge-clique, a contradiction. So we may assume now that this unique path of S intersecting paths of K is s_5. Using the same arguments as above, we may assume that c_7 is represented by a $\ulcorner$-path and c_8 by an $\llcorner$-path and P_{c_7}, P_{c_8} are the only paths of K using row x_i. We immediately conclude that P_{s_6} must intersect P_{c_7} on column y_j below row x_{i-1}, and P_{s_7} must intersect P_{c_8} on column y_j above row x_{i+1}. It follows from Proposition 1, that at least one of P_{s_1}, P_{s_2} intersects paths of K on column y_j, since $\{c_1, c_2, c_3, s_1, s_2\}$ induces a gem. But this is not possible since neither of them is comparable with one of s_6, s_7 (see Observation 4). Thus, G_5 does not belong to $[\llcorner, \ulcorner]_s$.

Lemma 5. $[\llcorner, \ulcorner, \urcorner]_s \subsetneq B_1$-$EPG_s$.

Proof. We clearly have $[\llcorner, \ulcorner, \urcorner]_s \subseteq B_1$-$EPG_s$. Consider the graph G_8 which belongs to B_1-EPG_s (see Fig. 8(a) and (b)). We will show that G_8 does not belong to $[\llcorner, \ulcorner, \urcorner]_s$. By contradiction suppose that $G_8 \in [\llcorner, \ulcorner, \urcorner]_s$. Assume first there exists a $[\llcorner, \ulcorner, \urcorner]$-representation of G_8, where the clique K induced by $\{c_1, \cdots, c_{10}\}$ is an edge-clique. Without loss of generality, we may assume that the base of K is vertical, say it goes from (x_i, y_j) to (x_{i+1}, y_j). Since $\{s_3, s_6, c_2, c_3, c_4\}$ induces a gem, it follows from Proposition 1 that at least one of P_{s_3}, P_{s_6} intersects paths of K on column y_j, say P_{s_3}. Similarly, since $\{s_9, s_{12}, c_7, c_8, c_9\}$ also induces a gem, we may assume that P_{s_9} intersects paths of K on column y_j. Since s_3 and s_9 are incomparable, one of these paths will be above row x_{i+1}, say P_{s_3}, and the other will be below row x_i, say P_{s_9} (see Observation 4). Now consider $P_{s_{12}}$. Since s_9 and s_{12} are incomparable and have a common neighbor, it follows from Observation 4 that $P_{s_{12}}$ must be above row x_{i+1}. But s_{12} has no common neighbour with s_3, so it follows from Observation 4 that $P_{s_{12}}$ must intersect all three paths P_{c_8}, P_{c_9} and $P_{c_{10}}$ on a same row, say row x_k, $k > i$, below P_{s_3}. Next consider s_{10} and s_{11}. Since they have each only one neighbour in K, and it is a common neighbour with s_{12}, it follows from the above that they must intersect this neighbour on the same row x_k. But since s_{10}, s_{11} are incomparable, they cannot intersect their neighbours on the same side of column y_j. So one will be to the right and the other to the left of column y_j (see Fig. 8(b)). Thus one of P_{c_9} and $P_{c_{10}}$ must be a ⌝-path and one must be a ⌜-path. The same reasoning can be done for s_6, s_4 and s_5 with the conclusion that one of P_{c_4} and P_{c_5} must be an ⌞-path and one must be a ⌟-path. But this contradicts the fact that $G_8 \in [$⌞, ⌜, ⌝$]$.

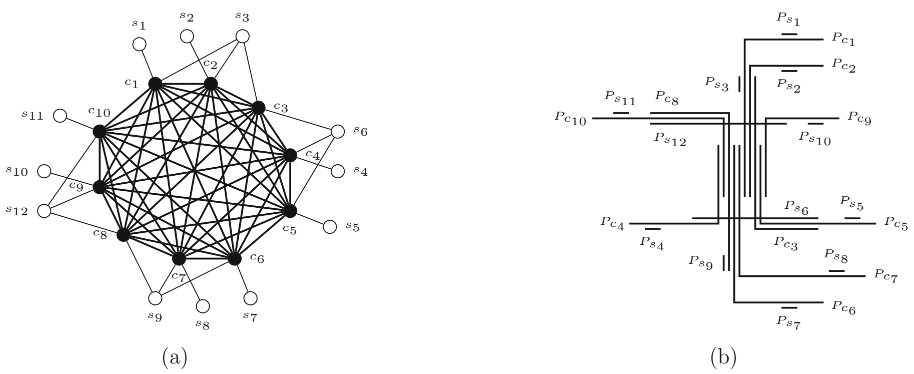


Fig. 8. The graph G_8 and a B_1-EPG representation of it.

Now assume there exists a $[$⌞, ⌜, ⌝$]$-representation of G_8 where the clique K is a claw-clique with center (x_i, y_j). Without loss of generality, we may assume that only paths of K use the grid-edges going from (x_{i-1}, y_j) to (x_{i+1}, y_j). Also, we may assume that all paths of K have a part lying on column y_j. It follows from Observation 4 that the S-vertices whose paths intersect paths of K on row x_i are pairwise comparable. Thus, we conclude that there can be at most two such vertices. First assume there are exactly two. Without loss of generality, we may assume that P_{s_1} and P_{s_3} intersect paths of K on row x_i (the proof is the same if two other paths of comparable S-vertices intersect paths of K on row x_i). Since s_2 and s_6 have both a common neighbour with s_3, their paths intersect paths of K on column y_j. Furthermore, s_2, s_6 are not comparable, so one of the paths P_{s_2}, P_{s_6} uses column y_j above (x_{i+1}, y_j) and the other uses column y_j below (x_{i-1}, y_j). Now $\{s_9, s_{12}, c_7, c_8, c_9\}$ induces a gem, thus it follows from Proposition 1 that at least one of the paths $P_{s_9}, P_{s_{12}}$ intersects paths of K on the column y_j. But this implies that s_9 or s_{12} is comparable with one of s_2 and s_6, a contradiction. So we may assume now that there is exactly one S-vertex whose path intersects paths of K on row x_i. We will distinguish two cases: this path represents an S-vertex of degree 3, or this paths represents an S-vertex of degree 1. First assume it is an S-vertex of degree 3, say, without loss of generality, s_3. Since s_1, s_2 and s_6 have a common neighbour with s_3, their paths must intersect paths of K on column y_j. But these three vertices are pairwise incomparable, a contradiction with Observation 4. Now assume, without loss of generality, that the path P_{s_1} is the unique path representing an S-vertex which intersects paths of K on row x_i. Similar to the proof of Lemma 4, every path of K with bend point (x_i, y_j) not intersecting any path of S on row x_i can be transformed into a vertical path (by deleting its horizontal part and extending it to (x_{i-1}, y_j) if it is an $\llcorner$-path and to (x_{i+1}, y_j) if it is a $\ulcorner$-path). Since s_1 has degree 1, it follows that exactly one path of K uses row x_i. Hence K is represented as an edge-clique, but this is impossible due to the above. Thus, G_8 is not in $[\llcorner, \ulcorner, \urcorner]_s$.

As for G_4 in the proof of Lemma 2, we can say that G_5 and G_8 are inclusion-wise minimal examples to show strictness of class inclusion for Lemmas 4 and 5, respectively. However, in both cases, we do not know whether there exist other examples with fewer vertices.

4 Split Graphs as $[\llcorner]$-Graphs

In this section, we characterise those split graphs that are in $[\llcorner]$. As already noticed in [7], gems and S-bulls play an important role with respect to the characterisation of split $[\llcorner]$-graphs.

Theorem 2. *Let G be a split graph with split partition (K, S). Then $G \in [\llcorner]$ if and only if there exist $S_1, S_2 \subseteq S$ such that:*

(a) each S_i for $i \in \{1, 2\}$ is a set of pairwise comparable vertices;
(b) for every gem in G with vertex set $\{c_1, s_1, c_2, s_2, c_3\}$ (see Fig. 2(a)), either $s_1 \in S_1$ or $s_2 \in S_1$;
(c) for every S-bull in G with vertex set $\{s_1, c_1, s_2, c_2, s_3\}$ (see Fig. 2(b)), at least one of s_1, s_2, s_3 belongs to S_1 or $s_2 \in S_2$.

Proof. Let G be a split graph with split partition (K, S). Assume that $G \in [\llcorner]$, and consider an $[\llcorner]$-representation of G. We define S_1 and S_2 as follows:

- S_1 is the set of vertices whose corresponding paths belong to the crown;
- S_2 is the set of vertices whose corresponding paths belong to the trunk.

It immediately follows from Observation 1 that each $S_i, i \in \{1, 2\}$ as defined above is a set of pairwise comparable vertices. Furthermore, it follows from Observation 2 that (b) is satisfied. Finally, (c) is an immediate consequence of Observation 3.

Conversely, let $G = (V, E)$ be a split graph with split partition (K, S), and assume that there exist $S_1, S_2 \subseteq S$ satisfying (a), (b) and (c). In addition, let us assume that we choose S_2 maximal with these properties. Let $S' = S \setminus (S_1 \cup S_2)$. Consider a partition $S'_1, S'_2, ..., S'_k$ of S' into non-empty sets such that $\forall\ i \neq j, N(S'_i) \cap N(S'_j) = \emptyset$ and k is maximal.

Claim 1: The vertices in S'_i, $i \in \{1, ..., k\}$, are pairwise comparable.

Let $s, s' \in S'_i$, for some $i \in \{1, ..., k\}$. Suppose that s, s' are not comparable. Denote by S''_i the vertices in S'_i that have a common neighbour with s. Then each vertex in S''_i is comparable to s. Indeed, let $u \in S''_i$. If u and s are not comparable, then there exist $c, c' \in K$ such that $sc, uc' \in E$ and $sc', uc \notin E$.

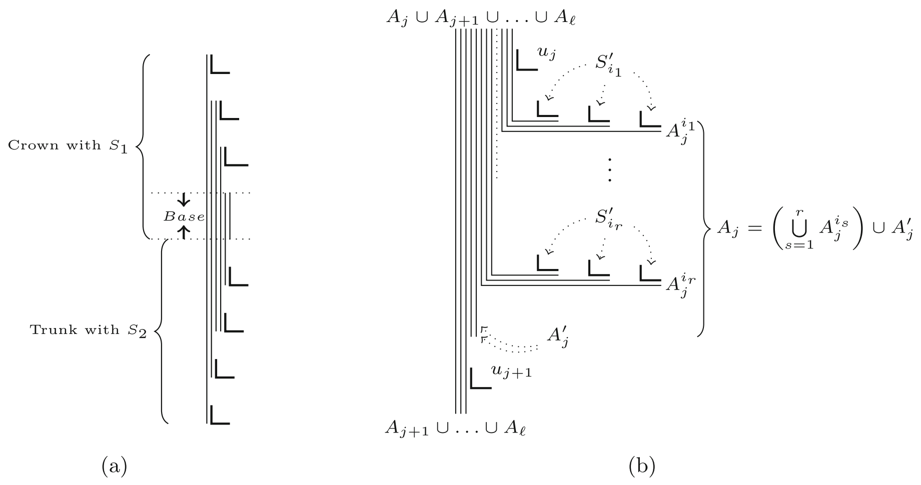
Since $u \in S''_i$, it follows that there exists c'' such that sc'', $uc'' \in E$. But then, $\{u, s, c, c', c''\}$ induces a gem, and hence (b) is not satisfied, a contradiction. So we conclude that $s' \notin S''_i$, since s, s' are incomparable. Now, assume there exist a vertex $u \in S''_i$ and a vertex $v \in S'_i \setminus S''_i$, $v \neq s$, that have a common neighbour c_1. Since $v \notin S''_i$, it follows that $sc_1 \notin E$. Then $\{s, u, v, c_1, c_2\}$ induces an S-bull, where c_2 is a common neighbour of s and u, and hence (c) is not satisfied, a contradiction. It follows from the above that we may partition S'_i into two sets, $S''_i \cup \{s\}$ and $S'_i \setminus (S''_i \cup \{s\})$ such that $N(S''_i \cup \{s\}) \cap N(S'_i \setminus (S''_i \cup \{s\})) = \emptyset$. But this contradicts the maximality of k. Therefore, s, s' are comparable. This proves Claim 1.

Let $S_2 = \{u_1, ..., u_\ell\}$ such that $N(u_\ell) \subseteq N(u_{\ell-1}) \subseteq ... \subseteq N(u_2) \subseteq N(u_1)$. Furthermore, let $A_0 = K \setminus N(u_1)$, for all $i \in \{1, ..., \ell - 1\}$ $A_i = N(u_i) \setminus N(u_{i+1})$ and $A_\ell = N(u_\ell)$.

Claim 2: There exists no set S'_i, $i \in \{1, ..., k\}$, such that $N(S'_i) \cap A_{j_1} \neq \emptyset$ and $N(S'_i) \cap A_{j_2} \neq \emptyset$, for $j_1 \neq j_2$ and $j_1, j_2 \in \{0, 1, \dots, \ell\}$.

Let S'_i be such that $x \in N(S'_i) \cap A_{j_1}$ and $y \in N(S'_i) \cap A_{j_2}$, for $j_1 \neq j_2$ and $j_1, j_2 \in \{0, 1, \dots, \ell\}$. Without loss of generality, we may assume that $j_1 < j_2$ and that j_1 is chosen smallest with the property that $N(S'_i) \cap A_{j_1} \neq \emptyset$. Let u be a dominant vertex in S'_i, i.e. $N(u) = N(S'_i)$. Consider vertex $u_{j_1+1} \in S_2$. Notice that x and u_{j_1+1} are not adjacent. Hence, if there exists a vertex $z \in K$ which is adjacent to u_{j_1+1} and non-adjacent to u, then $\{u, u_{j_1+1}, x, y, z\}$ induces a gem, and hence (b) is not satisfied, a contradiction. Thus, u dominates u_{j_1+1}. Since u_{j_1+1} dominates u_j, for $j = j_1 + 2, \dots, \ell$, we conclude that u actually dominates u_j, for $j = j_1 + 1, \dots, \ell$. If $j_1 = 0$, we obtain that u dominates all vertices in S_2, and thus we may add u to S_2 (and (a), (b), (c) would still be satisfied), which contradicts the maximality of S_2. So we may assume that $j_1 > 0$. Notice that u is dominated by every vertex u_j, with $j \in \{1, \dots, j_1\}$, since j_1 is chosen smallest with the property that $N(S'_i) \cap A_{j_1} \neq \emptyset$. Hence, we may again add u to S_2 (and (a), (b), (c) would still be satisfied), which contradicts the maximality of S_2. This proves Claim 2.

We will construct an $[\llcorner]$-representation of G as follows. We start with the base, which, without loss of generality, we may assume vertical. Next, we extend the paths of the base and add all vertices of S_1 in the crown and all vertices of S_2 in the trunk (see Fig. 9(a)). This is possible since the vertices in S_1 (resp. S_2) are pairwise comparable. Notice that currently each path P_c, for $c \in A_j$, $j \in \{1, \dots, \ell\}$, has its lower endpoint below P_{u_j} and above $P_{u_{j+1}}$, and each path P_c, for $c \in A_0$, has its lower endpoint above P_{u_1}. Consider a set A_j, $j \in \{0, 1, \dots, \ell\}$ as well as all sets among $S'_1, \dots, S'_k$ which have neighbours in A_j, say $S'_{i_1}, \dots, S'_{i_r}$. It follows from Claim 2 and the fact that $N(S'_i) \cap N(S'_l) = \emptyset$ for all $i, l \in \{1, \dots, k\}$, $i \neq l$, that we may partition the vertices of A_j into sets $A_j^{i_1}, \dots, A_j^{i_r}, A'_l$ such that $N(S'_{i_s}) = A_j^{i_s}$, for $s = 1, \dots, r$ and the vertices of A'_j have no neighbours in S'. Since each set S'_i for $i \in \{1, ..., k\}$ contains pairwise comparable vertices (see Claim 1), we may now represent the vertices of each set



Crown with S_1
Base
Trunk with S_2
(a)
$A_j \cup A_{j+1} \cup \ldots \cup A_\ell$
u_j
S'_{i_1}
$A_j^{i_1}$
S'_{i_r}
$A_j^{i_r}$
$A_j = \left(\bigcup_{s=1}^{r} A_j^{i_s}\right) \cup A'_j$
A'_j
u_{j+1}
$A_{j+1} \cup \ldots \cup A_\ell$
(b)

Fig. 9. Illustration of Theorem 2.

S'_{i_s} on a separate branch formed by the horizontal parts of the paths P_c, with $c \in A_j^{i_s}$, for $s = 1, \ldots, r$ (see Fig. 9(b)).

Notice that the previous characterisation does not imply that graphs in $[\llcorner]_s$ can be recognised in polynomial time. This still remains open.

In [7], the authors state a conjecture concerning the characterisation of the class $[\llcorner]_s$ by a family of forbidden induced subgraphs. Here, we will show that the conjecture is wrong by presenting an additional forbidden induced subgraph that was not mentioned in their list (and which is not contained in any of their forbidden graphs as induced subgraph), using Theorem 2. Consider the graph H shown in Fig. 10. We obtain the following.

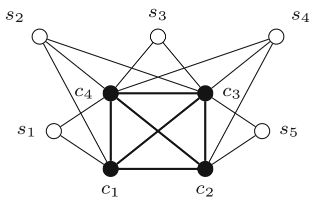


Fig. 10. The graph H.

Lemma 6. *The graph H is not in $[\llcorner]_s$.*

Proof. Suppose by contradiction that $H \in [\llcorner]_s$. Since $\{c_1, c_3, c_4, s_1, s_3\}$ induces a gem, it follows from Theorem 2, that either s_1 or s_3 belong to S_1.

First assume that $s_1 \in S_1$. Hence $s_3 \notin S_1$. Now $\{c_2, c_3, c_4, s_3, s_5\}$ also induces a gem. It follows again from Theorem 2 and the fact that $s_3 \notin S_1$ that $s_5 \in S_1$. So both s_1, s_5 belong to S_1. But they are incomparable, a contradiction.

So we may assume now that $s_3 \in S_1$ and $s_1 \notin S_1$. The vertex set $\{c_1, c_3, c_4, s_1, s_4\}$ induces a gem. Thus, it follows from Theorem 2 and the fact that $s_1 \notin S_1$ that $s_4 \in S_1$. Similarly, the vertices $\{c_2, c_3, c_4, s_2, s_5\}$ induces a gem. So according to Theorem 2, either s_2 or s_5 belongs to S_1. But s_2, s_4 are incomparable and s_5, s_3 are incomparable. Thus, we obtain again a contradiction since S_1 is a set of pairwise comparable vertices.

Note that one can easily check that the graph H is a minimal forbidden induced subgraph by removing each vertex separately and applying Theorem 2.

5 Conclusion

In this paper, we were interested in split graphs as edge intersection graphs of single bend paths on a grid. We presented a characterisation of this graph class using the notions of gems and S-bulls. Our characterisation allowed us to disprove a conjecture by Cameron et al. stating that this class can be characterised by a list of 9 forbidden induced subgraphs [7]. Notice that, even though we only gave here a single additional forbidden induced subgraph, we actually managed to detect 20 new ones so far. Furthermore, we investigated some subclasses of split B_1-EPG graphs for which only a subset of the four possible shapes are allowed. We presented the complete set of inclusion relationships between these graph families.

Our characterisation mentioned above does not immediately lead to a polynomial-time recognition algorithm. Thus, it is still open whether split B_1-EPG graphs can be recognised in polynomial time or not. Furthermore, it would be interesting to obtain a characterisation of chordal B_1-EPG graphs.

In [7], the authors present a characterisation of gem-free (resp. S-bull-free) graphs that are in $[\llcorner]_s$. We managed to generalise these results to gem-free (resp. S-bull-free) graphs that are in $[P]_s$, for any subset P of $\{\llcorner, \urcorner, \ulcorner, \lrcorner\}$. All these graph classes can be recognised in polynomial time. Due to space constraints, we were not able to include these results in the present paper.

Acknowledgements. The first author has been supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under 2214-A Doctoral Research Program Grant during his stay in Switzerland. This work was done while the first and second authors visited the University of Fribourg, Switzerland. The support of the institution is gratefully acknowledged.

References

1. Alcón, L., Bonomo, F., Durán, G., Gutierrez, M., Mazzoleni, M.P., Ries, B., Valencia-Pabon, M.: On the bend number of circular-arc graphs as edge intersection graphs of paths on a grid. Discrete Appl. Math. **234**, 12–21 (2017). https://doi.org/10.1016/j.dam.2016.08.004
2. Asinowski, A., Ries, B.: Some properties of edge intersection graphs of single-bend paths on a grid. Discrete Math. **312**, 427–440 (2012)

3. Asinowski, A., Suk, A.: Edge intersection graphs of systems of paths on a grid with a bounded number of bends. Discrete Appl. Math. **157**(14), 3174–3180 (2009)
4. Bield, T., Stern, M.: Edge-intersection graphs of k-bend paths in grids. Discrete Math. Theor. Comput. Sci. **12:1**, 1–12 (2010)
5. Bonomo, F., Mazzoleni, M.P., Stein, M.: Clique coloring B_1-EPG graphs. Discrete Math. **340**(5), 1008–1011 (2017)
6. Bougeret, M., Bessy, S., Gonçalves, D., Paul, C.: On independent set on B1-EPG graphs. In: Sanità, L., Skutella, M. (eds.) WAOA 2015. LNCS, vol. 9499, pp. 158–169. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28684-6_14
7. Cameron, K., Chaplick, S., Hoang, C.T.: Edge intersection graphs of L-shaped paths in grids. Discrete Appl. Math. **210**, 185–194 (2016)
8. Cohen, E., Golumbic, M.C., Ries, B.: Characterizations of cographs as intersection graphs of paths on a grid. Discrete Appl. Math. **178**, 46–57 (2014)
9. Diestel, R.: Graph Theory. Springer, Heidelberg (2005)
10. Epstein, D., Golumbic, M.C., Morgenstern, G.: Approximation algorithms for B_1-EPG graphs. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 328–340. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_29
11. Francis, M.C., Lahiri, A.: VPG and EPG bend-numbers of halin graphs. Discrete Appl. Math. **215**, 95–105 (2016)
12. Golumbic, M., Lipshteyn, M., Stern, M.: Edge intersection graphs of single bend paths on a grid. Networks **54**, 130–138 (2009)
13. Heldt, D., Knauer, K., Ueckerdt, T.: Edge-intersection graphs of grid paths: the bend-number. Discrete Appl. Math. **167**, 144–162 (2014)
14. Heldt, D., Knauer, K., Ueckerdt, T.: On the bend-number of planar and outerplanar graphs. Discrete Appl. Math. **179**, 109–119 (2014)
15. Pergel, M., Rzążewski, P.: On edge intersection graphs of paths with 2 bends. In: Heggernes, P. (ed.) WG 2016. LNCS, vol. 9941, pp. 207–219. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53536-3_18
16. West, D.B.: Introduction to Graph Theory. Prentice-Hall, Upper Saddle River (1996)

Efficient Algorithms for Computing a Minimal Homology Basis

Tamal K. Dey, Tianqi Li(✉), and Yusu Wang

Department of Computer Science and Engineering,
The Ohio State University, Columbus, USA
{tamaldey,yusu}@cse.ohio-state.edu, li.6108@osu.edu

Abstract. Efficient computation of shortest cycles which form a homology basis under $\mathbb{Z}_2$-additions in a given simplicial complex $\mathcal{K}$ has been researched actively in recent years. When the complex $\mathcal{K}$ is a weighted graph with n vertices and m edges, the problem of computing a shortest (homology) cycle basis is known to be solvable in $O(m^2n/\log n + n^2m)$-time. Several works [1,2] have addressed the case when the complex $\mathcal{K}$ is a 2-manifold. The complexity of these algorithms depends on the rank g of the one-dimensional homology group of $\mathcal{K}$. This rank g has a lower bound of $\Theta(n)$, where n denotes the number of simplices in $\mathcal{K}$, giving an $O(n^4)$ worst-case time complexity for the algorithms in [1,2]. This worst-case complexity is improved in [3] to $O(n^\omega + n^2g^{\omega-1})$ for general simplicial complexes where $\omega < 2.3728639$ [4] is the matrix multiplication exponent. Taking $g = \Theta(n)$, this provides an $O(n^{\omega+1})$ worst-case algorithm. In this paper, we improve this time complexity. Combining the divide and conquer technique from [5] with the use of annotations from [3], we present an algorithm that runs in $O(n^\omega + n^2g)$ time giving the first $O(n^3)$ worst-case algorithm for general complexes. If instead of minimal basis, we settle for an approximate basis, we can improve the running time even further. We show that a 2-approximate minimal homology basis can be computed in $O(n^\omega\sqrt{n \log n})$ expected time. We also study more general measures for defining the minimal basis and identify reasonable conditions on these measures that allow computing a minimal basis efficiently.

1 Introduction

Many applications in science and engineering require computing "features" in a shape that is finitely represented by a simplicial complex. These features sometimes include topological features such as "holes" and "tunnels" present in the shape. A concise definition of these otherwise vague notions can be obtained by considering homology groups and their representative cycles. In particular, a one-dimensional homology basis, that is, a set of independent cycles in the 1-skeleton of the input simplicial complex whose homology classes form a basis for the first homology group, can be taken as a representative of the "holes" and "tunnels" present in the shape. However, instead of any basis, one would like to

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 376–398, 2018.
https://doi.org/10.1007/978-3-319-77404-6_28

have a homology basis whose representative cycles are small under some suitable metric, thus bringing the 'geometry' into picture along with topology.

When the input complex is a graph with n vertices and m edges, the homology basis coincides with what is called the cycle basis and its minimality is measured with respect to the total weights of the cycles assuming non-negative weights on the edges. A number of efficient algorithms have been designed to compute such a minimal cycle basis for a weighted graph [1,5–8]. The best known algorithm for this case runs in $O(m^2 n/\log n + n^2 m)$ [8].

When the input is a simplicial complex, one dimensional homology basis is determined by the simplices of dimension up to 2. Thus, without loss of generality, we can assume that the complex has dimension at most 2, that is, it consists of vertices, edges, and triangles. The 1-skeleton of the complex is a graph (weighted if the edges are). Therefore, one can consider a minimal cycle basis in the 1-skeleton. However, the presence of triangles makes some of these basis elements to be trivial in the homology basis. Therefore, the computation of the minimal homology basis in a simplicial complex differs from the minimal cycle basis in a graph. In this paper, we show that the efficient algorithms of [5] for computing *a minimal cycle basis* can be adapted to computing *a minimal homology basis* in a simplicial complex (by combining with an algorithm [3] to compute the so-called annotations). In the process we improve the current best time complexity bound for computing a minimal homology basis and also extend these results to more generalized measures.

More specifically, for the special case of a combinatorial 2-manifold with weights on the edges, Erickson and Whittlesey [2] gave an $O(n^2 \log n + gn^2 + g^3 n)$-time algorithm to compute a minimal homology basis where n is the total number of simplices and g is the rank of the first homology group. Dey et al. [9] and Chen and Friedman [10] generalized the results above to arbitrary simplicial complexes. Busaryev et al. [3] improved the running time of this generalization from $O(n^4)$ [9] to $O(n^\omega + n^2 g^{\omega-1})$ where $\omega < 2.3728639$ [4] is the matrix multiplication exponent. This gives the best known $O(n^{1+\omega})$ worst-case time algorithm when $g = \Theta(n)$. In Sect. 3, combining the divide and conquer approach of [5] with the use of annotations [3], we develop an improved algorithm to compute a minimal 1-dimensional homology basis for an arbitrary simplicial complex in only $O(n^2 g + n^\omega)$ time. Considering $g = \Theta(n)$, this gives the first $O(n^3)$ worst-case time algorithm for the problem.

We can further improve the time complexity if we allow for approximation. An algorithm to compute a 2-approximate minimal homology basis is given in Sect. 4 running in $O(n^\omega \sqrt{n \log n})$ expected time.

All of the above algorithms operate by computing a set of candidate cycles that necessarily includes at least one minimal homology basis and then selecting one of these minimal bases. The standard proof [2] of the fact that the candidate set includes a minimal basis uses the specific distance function based on the shortest path metric and a size function that assigns total weight of the edges in a cycle as its size. In Sect. 5, we identify general conditions for the distance and size function so that the divide and conquer algorithm still works without degrading in time complexity. This allows us to consider distance function beyond the shortest path metric and the size function beyond the total weight of edges as we illustrate with two examples. Specifically, we can now compute a minimal homology basis whose size is induced by a general map $F : \mathcal{K} \to Z$ for any metric space Z.

2 Background and Notations

In this paper, we are interested in computing a minimal basis for the 1-dimensional homology group of a simplicial complex over the field $\mathbb{Z}_2$. In this section we briefly introduce some relevant concepts here; the details appear in standard books on algebraic topology such as [11].

Homology. Let $\mathcal{K}$ be a connected simplicial complex. A d-chain c is a formal sum, $c = \sum a_i \sigma_i$ where the σ_is are the d-simplices of $\mathcal{K}$ and the a_is are the coefficients with $a_i \in \mathbb{Z}_2$. We use C_d to denote the group of d-chains which is formed by the set of d-chains together with the addition. Note that there is a one-to-one correspondence between the chain group C_d and the family of subsets of $\mathcal{K}_d$ where $\mathcal{K}_d$ is the set of all d-simplices. Thus C_d is isomorphic to the space $(\mathbb{Z}_2)^{n_d}$ where n_d is the number of d-simplices in $\mathcal{K}$. Naturally all d-simplices in $\mathcal{K}$ form a basis of C_d in which the i-th bit of the coordinate vector of a d-chain indicates whether the corresponding d-simplex appears in the chain.

The boundary of a d-simplex is the sum of all its $(d-1)$-faces. This can be interpreted and extended to a d-chain as a *boundary map* $\partial_d : \mathsf{C}_d \to \mathsf{C}_{d-1}$, where the boundary of a chain is defined as the sum of the boundaries of its simplices. A d-cycle c is a d-chain with empty boundary, $\partial_d c = 0$. Since ∂_d commutes with addition, we have the group of d-cycles, Z_d, which is the kernel of ∂_d, $\mathsf{Z}_d := ker\partial_d$. A d-boundary c is a d-chain that is the boundary of a $(d+1)$-chain, $c = \partial_{d+1} b$ for some $b \in \mathsf{C}_{d+1}$. The group of d-boundaries B_d is the image of ∂_{d+1}, that is, $\mathsf{B}_d := im\partial_{d+1}$. Notice that B_d is a subgroup of Z_d. Hence we can consider the quotient $\mathsf{Z}_d/\mathsf{B}_d$ which constitutes the d-dimensional homology group denoted as H_d. Each element in H_d, called a homology class, is an equivalence class of d-cycles whose difference is always in B_d. Two cycles are said to be *homologous* if they are in the same homology class.

Under $\mathbb{Z}_2$ coefficients, the groups C_d, Z_d, B_d and H_d are all vector spaces. A basis of a vector space is a set of vectors of minimal cardinality that generates the entire vector space. We are concerned with the homology bases of H_d and particularly in H_1 (more formally below). We use $L = rank(\mathsf{Z}_1)$ to denote the dimension of vector space Z_1 and use $g = rank(\mathsf{H}_1)$ to denote the 1-*st Betti number* of $\mathcal{K}$, which is the dimension of vector space H_1.

- A set of cycles $C_1, \cdots, C_L$, with $L = rank(\mathsf{Z}_1)$, that generates the cycle space Z_1 is called its *cycle basis*.
- For any 1-cycle c, let $[c]$ denote its homology class. A set of homology classes $\{[C_1], \ldots, [C_g]\}$ that constitutes a basis of H_1 is called a *homology basis*. For simplicity, we also say a set of cycles $\{C_1, C_2, \cdots, C_g\}$ is a homology basis

if their corresponding homology classes $[C_1], [C_2], \cdots, [C_g]$ form a basis for $\mathsf{H}_1(\mathcal{K})$.
- Let $\mu : \mathsf{Z}_1 \to \mathbb{R}^+ \cup \{0\}$ be a size function that assigns a non-negative weight to each cycle $C \in \mathsf{Z}_1$. A cycle or homology basis $C_1, \cdots, C_l$ is called *minimal* if $\sum_{i=1}^{l} \mu(C_i)$ is minimal among all bases of Z_1 ($l = L$) or $\mathsf{H}_1(\mathcal{K})$ ($l = g$) respectively.

Annotation. To compute a minimal homology basis of a simplicial complex $\mathcal{K}$, it is necessary to have a way to represent and distinguish homology classes of cycles. Annotated simplices have been used for this purpose in earlier works: For example, Erickson and Wittlesey [2] and Borradaile et al. [1] used them for computing optimal homology cycles in surface embedded graphs. Here we use a version termed as *annotation* from [3] which gives an algorithm to compute them in matrix multiplication time for general simplicial complexes. An annotation for a d-simplex is a g-bit binary vector, where $g = rank(\mathsf{H}_d(\mathcal{K}))$. The annotation of a cycle z, which is the sum of annotations of all simplices in z, provides the coordinate vector of the homology class of z in a pre-determined homology basis. More formally,

Definition 2.1 (Annotation). *Let $\mathcal{K}$ be a simplicial complex and $\mathcal{K}_d$ be the set of d-simplices in $\mathcal{K}$. An annotation for d-simplices is a function $a : \mathcal{K}_d \to (\mathbb{Z}_2)^g$ with the following property: any two d-cycles z and z' are homologous if and only if*

$$\sum_{\sigma \in z} a(\sigma) = \sum_{\sigma \in z'} a(\sigma)$$

Given an annotation a, the annotation of any d-cycle z is defined by $a(z) = \sum_{\sigma \in z} a(\sigma)$.

Proposition 2.1 ([3]). *There is an algorithm that annotates the d-simplices in a simplicial complex with n simplices in $O(n^{\omega})$ time.*

3 Minimal Homology Basis

In this section, we describe an efficient algorithm to compute a minimal homology basis of the 1-dimensional homology group $\mathsf{H}_1(\mathcal{K})$. The algorithm uses the divide and conquer technique from [5] where they compute a minimal *cycle* basis in a weighted graph. The authors in [1] adapted it for computing optimal homology basis in surface embedded graphs. We adapt it here to simplicial complexes using edge annotations [3].

More specifically, let $\mathcal{K}$ be a simplicial complex with n simplices – Since we are only interested in 1-dimensional homology basis, it is sufficient to consider all simplices with dimension up to 2, namely vertices, edges, and triangles. Hence we assume that $\mathcal{K}$ contains only simplices of dimension at most 2. Assume that the edges in $\mathcal{K}$ are weighted with non-negative weights. Given any homology basis $\{C_1, \ldots, C_g\}$ where $g = rank(\mathsf{H}_1(\mathcal{K}))$, we define the *size* $\mu(C)$ *of a* *cycle* $C \in Z_1(\mathcal{K})$ as the *total weights* of its edges. As defined in Sect. 2, the problem of computing a minimal homology basis of H_1 is now to find a basis $\mathcal{C} = \{C_1, C_2, \cdots, C_g\}$ such that the sum of $\sum_{i=1}^{g} \mu(C_i)$ is the smallest.

The high-level algorithm to compute such a minimal homology basis of H_1 group proceeds as follows. First, we need to annotate all 1-simplices implemented by the algorithm of [3]. Then we compute a candidate set of cycles which includes a minimal homology basis. At last, we extract such a minimal homology basis from the candidate set.

Candidate set. We now describe the step to compute a candidate set $\mathcal{G}$ of cycles that contains a minimal homology basis. We use the shortest path tree approach which dates back to Horton's algorithm for a minimal cycle basis of a graph [7]. It was also applied in other earlier works, e.g. [2,9]. We first generate a candidate set $\mathcal{G}(p)$ for every vertex $p \in \mathit{vert}(\mathcal{K})$, where $\mathit{vert}(\mathcal{K})$ is the set of vertices of $\mathcal{K}$. Then we take the union of all $\mathcal{G}(p)$ and denote as $\mathcal{G}$, i.e. $\mathcal{G} = \cup_{p \in \mathit{vert}(\mathcal{K})} \mathcal{G}(p)$. To compute $\mathcal{G}(p)$, first we construct a shortest path tree T_p rooted at p. Let $\Pi_p(u, v)$ denote the unique path connecting two vertices u and v in T_p. Then each nontree edge $e = (u, v)$ generates a cycle $C(p, e) = e \circ \Pi_p(u, v)$. The union of all such cycles constitutes the candidate set of the vertex p, i.e. $\mathcal{G}(p) = \cup_{e \in \mathit{edge}(\mathcal{K}) \setminus E_p} C(p, e)$ where E_p is the set of tree edges in T_p. Note that the number of cycles in $\mathcal{G}(p)$ is $O(n)$ for each vertex $p \in \mathit{vert}(\mathcal{K})$. Hence there are $O(n^2)$ candidate cycles in $\mathcal{G}$ in total. They, together with their sizes, can be computed in $O(n^2 \log n)$ time.

Proposition 3.1 **([2,9]).** *The candidate set $\mathcal{G}$ has $O(n^2)$ cycles and admits a minimal homology basis.*

3.1 Computing a Minimal Homology Basis

What remains is to compute a minimal homology basis from the candidate set $\mathcal{G}$. To achieve it, we modify the divide and conquer approach from [5] which improved the algorithm of [6] for computing a minimal cycle basis of a graph with non-negative weights.

This approach uses an auxiliary set of support vectors [5] that helps select a minimal homology basis from a larger set containing at least one minimal basis; in our case, this larger set is $\mathcal{G}$.

A support vector S is a vector in the space of g-dimensional binary vectors $\mathcal{S} = \{0, 1\}^g$. The use of support vectors along with annotations requires us to perform more operations without increasing the complexity of the divide and conquer approach. Let $a(C)$ denote the annotation of a cycle C. First, we define the function:

$$m : \mathcal{S} \times \mathcal{G} \to \{0, 1\} \text{ with } m(S, C) = \langle S, a(C) \rangle$$
$$\text{where } \langle \cdot, \cdot \rangle \text{ is the inner product over } \mathbb{Z}_2.$$

We say a cycle C is *orthogonal* to a support vector S_i if $m(S_i, C) = 0$ and is *non-orthogonal* if $m(S_i, C) = 1$. We would choose cycles $C_1, \cdots, C_g$ iteratively from a set guaranteed to contain a minimal homology basis and add them to the minimal homology basis. During the procedure, the algorithm always maintains a set of support vectors $S_1, S_2, \cdots, S_g$ with the following properties:

(1) $S_1, S_2, \cdots, S_g$ form a basis of $\{0,1\}^g$.
(2) If $C_1, C_2, \cdots, C_{i-1}$ have already been computed, $m(S_i, C_j) = 0$, $j < i$.

Suppose that in addition to properties (1) and (2), we have the following additional condition to choose C_is, then the set $C_1, C_2, \ldots, C_g$ constitutes a minimal homology basis.

(3) If $C_1, C_2, \cdots, C_{i-1}$ have already been computed, C_i is chosen so that C_i is the shortest cycle with $m(S_i, C_i) = 1$.

If we keep the same support vectors, after we select a new cycle C_i, $m(S_{i+1}, C_i) = 0$ may not hold which means the property (2) may not hold. Therefore, we update the support vectors $S_{i+1}, \cdots, S_g$ after computing C_i so that the orthogonality condition (2) holds. If chosen with condition (3), the cycle C_i becomes independent of the cycles previously chosen as stated below:

Claim 3.1. *For any $i \leq g$, if property (1) and (2) hold, then for any cycle C with $m(S_i, C) = 1$, $[C]$ is independent of $[C_1], [C_2], \cdots, [C_{i-1}]$.*

Proof. By property (2), $\forall j < i, m(S_i, C_j) = 0$. If $[C]$ is not independent of $[C_1], [C_2], \cdots, [C_{i-1}]$, then the annotation $a(C)$ of the cycle C can be written as $a(C) = \sum_{j<i} \alpha_j a(C_j)$, where $\alpha_j \in \{0,1\}$ and at least one $\alpha_j = 1, j < i$. Since $m(S_i, C_i) = 1$, we have $\sum_{j<i} \alpha_j m(S_i, C_j) = 1$. It follows that there exists at least one C_j, $j < i$, with $m(S_i, C_j) = 1$, which contradicts with property (2). Therefore, $[C]$ is independent of $[C_1], [C_2], \cdots, [C_{i-1}]$. □

The following theorem guarantees that the above three conditions suffice for a minimal homology basis. Its proof is almost the same as the proof of [5, Theorem 1] (which draws upon the idea of [6]).

Theorem 3.1. *The set $\{C_1, C_2, \cdots, C_g\}$ computed by maintaining properties (1), (2) and (3) is a minimal homology basis.*

Taking advantage of the above theorem, we aim to compute a homology basis iteratively while maintaining conditions (1), (2), and (3).

Maintaining support vectors and computing shortest cycles. Now we describe the algorithm CycleBasis($\mathcal{G}$) (given in Algorithm 1) that computes a minimal homology basis. In this algorithm, we first initialize each support vector S_i so that only the i-th bit is set to 1. Then the main computation is done by calling the procedure ExtendBasis$(1, g)$.

Here the procedure ExtendBasis(i, k) (Algorithm 2) is recursive which extends the current partial basis $\{C_1, \cdots, C_{i-1}\}$ by adding k new cycles. It modifies a divide and conquer approach of [5] to maintain properties (1), (2), and (3).

Algorithm 1. Computing a minimal Basis

```
1: procedure CycleBasis($\mathcal{G}$)
2:     for $i \leftarrow 1$ to $g$ do
3:         Initialize $S_i \leftarrow \{e_i\}$, which means that the $i$-th bit of $S_i$ is 1 while others are 0
4:     end for
5:     ExtendBasis($1, g$) to get a minimal homology basis $\{C_1, \cdots, C_g\}$
6: end procedure
```

It calls a routine Update to maintain orthogonality using annotations. For choosing the shortest cycle satisfying condition (3), it calls ShortestCycle(S_i) in the base case ($k = 1$)(See line 3 of Algorithm 2). We describe the recursion and the base case below.

Algorithm 2. Extend the Basis by k elements

```
1: procedure ExtendBasis($i$, $k$)
2:     if $k = 1$ then
3:         Call ShortestCycle($S_i$) to compute the shortest cycle $C_i$ which is non-orthogonal to $S_i$
4:     else
5:         Call ExtendBasis($i$, $\lfloor k/2 \rfloor$) to extend the homology basis by $\lfloor k/2 \rfloor$ elements. After calling, $S_i, \ldots, S_{i+\lfloor k/2 \rfloor -1}$ will be updated.
6:         Call Update($i$, $k$) to update the support vectors $\{S_{i+\lfloor k/2 \rfloor}, \ldots, S_{i+k-1}\}$ using $\{S_i, \ldots, S_{i+\lfloor k/2 \rfloor -1}\}$ and update the value $m(S_j, e)$ for $i + \lfloor k/2 \rfloor \le j \le i+k-1$ and every edge $e$.
7:         Call ExtendBasis($i + \lfloor k/2 \rfloor$, $\lceil k/2 \rceil$) to extend the cycle basis by $\lceil k/2 \rceil$ elements
8:     end if
9: end procedure
```

Recursion. At the high level, the procedure ExtendBasis(i, k) recurses on k by first calling itself to obtain the next $\lfloor k/2 \rfloor$ cycles in the minimal homology basis in which the support vectors $S_i, S_{i+1}, \cdots, S_{i+\lfloor k/2 \rfloor -1}$ are updated. Then it calls the procedure Update(i, k) to maintain the orthogonality property (2). It uses the already updated support vectors $S_i, \cdots, S_{i+\lfloor k/2 \rfloor -1}$ to update $\{S_{i+\lfloor k/2 \rfloor}, \ldots, S_{i+k-1}\}$ so that $m(S_l, C_j) = 0, \forall j < i + \lfloor k/2 \rfloor, i + \lfloor k/2 \rfloor \le l \le i + k - 1$. At last the procedure ExtendBasis(i, k) calls itself ExtendBasis($i + \lfloor k/2 \rfloor, \lceil k/2 \rceil$) to extend the basis by $\lceil k/2 \rceil$ elements.

We describe Update(i, k) and spare giving its pseudocode. Let $\{\hat{S}_{i+\lfloor k/2 \rfloor}, \ldots, \hat{S}_{i+k-1}\}$ denote the desired output vectors after the update. To ensure the property (1) and (2), we will enforce that the vector $\hat{S}_j$ is of the form $\hat{S}_j = S_j + \sum_{t=1}^{\lfloor k/2 \rfloor} \alpha_{jt} S_{i+t-1}$ where $i + \lfloor k/2 \rfloor \le j \le i + k - 1$. We just need to determine the coefficients $\alpha_{j1}, \ldots, \alpha_{j\lfloor k/2 \rfloor}$ so that $m(\hat{S}_j, C_t) = 0$ where $i + \lfloor k/2 \rfloor \le j \le i + k - 1$ and $i \le t \le i + \lfloor k/2 \rfloor - 1$. We will also compute $m(S_j, e)$

for $i + \lfloor k/2 \rfloor \leq j \leq i + k - 1$ and every edge e where $m(S_j, e)$ is defined as the standard inner product of S_j and $a(e)$ under $\mathbb{Z}_2$, which is important later when we compute the shortest cycle orthogonal to a support vector S in the procedure ShortestCycle(S).

Now let

$$X = \begin{pmatrix} S_i \\ S_{i+1} \\ \vdots \\ S_{i+\lfloor k/2 \rfloor - 1} \end{pmatrix} \cdot \left(a(C_i)^T \; a(C_{i+1})^T \; \cdots \; a(C_{i+\lfloor k/2 \rfloor - 1})^T \right)$$

$$Y = \begin{pmatrix} S_{i+\lfloor k/2 \rfloor} \\ S_{i+\lfloor k/2 \rfloor + 1} \\ \vdots \\ S_{i+k-1} \end{pmatrix} \cdot \left(a(C_i)^T \; a(C_{i+1})^T \; \cdots \; a(C_{i+\lfloor k/2 \rfloor - 1})^T \right),$$

where recall that g-bit vector $a(C)$ is the annotation of a cycle C. Let A denote a $\lceil k/2 \rceil \times \lfloor k/2 \rfloor$ matrix where row j contains the bit $\alpha_{j+i+\lfloor k/2 \rfloor -1,1}, \cdots, \alpha_{j+i+\lfloor k/2 \rfloor -1,\lfloor k/2 \rfloor}$. It is not difficult to see that $AX + Y = 0$, and that X is invertible, which means that $A = -YX^{-1} = YX^{-1}$ since the computations are under $\mathbb{Z}_2$.

The next step is to update the value $m(S_j, e)$ to $m(\hat{S}_j, e)$ for every edge e in $\mathcal{K}$, and $i + \lfloor k/2 \rfloor \leq j \leq i+k-1$. Note that the coefficients α_{jt} are now known and the updated vectors are $\hat{S}_j = S_j + \sum_{t=1}^{\lfloor k/2 \rfloor} \alpha_{jt} S_{i+t-1}$, $i + \lfloor k/2 \rfloor \leq j \leq i + k - 1$. Thus for every edge e, $m(\hat{S}_j, e) = m(S_j + \sum_{t=1}^{\lfloor k/2 \rfloor} \alpha_{jt} S_{i+t-1}, e) = m(S_j, e) + \sum_{t=1}^{\lfloor k/2 \rfloor} \alpha_{jt} m(S_{i+t-1}, e)$, $i + \lfloor k/2 \rfloor \leq j \leq i+k-1$. Let n_1 be the number of edges in $\mathcal{K}$ and U be the $\lceil k/2 \rceil \times n_1$ matrix where its (t, j) entry is $m(\hat{S}_{i+\lfloor k/2 \rfloor + t - 1}, e_j)$. Set $W = [A|I]$ where I is the $\lceil k/2 \rceil \times \lceil k/2 \rceil$ identity matrix. Let Z be a $k \times n_1$ matrix whose (s, t) entry is $m(S_{i+s-1}, e_t)$. Thus we have $U = WZ$. Since the $\lceil k/2 \rceil \times k$ matrix W and $k \times n_1$ matrix Z are already known, the matrix U can be computed in $O(nk^{\omega-1})$ time by chopping Z to n_1/k number of $k \times k$ submatrices and performing $O(n_1/k)$ matrix multiplications of two $O(k) \times O(k)$ size matrices. After that, $m(\hat{S}_j, e)$ can be easily retrieved from the matrix U in constant time.

Base case for selecting a shortest cycle. We now implement the procedure ShortestCycle(S_i) for the base case to compute the shortest cycle C_i non-orthogonal to S_i, i.e. the shortest cycle C_i satisfying $m(S_i, C_i) = 1$. We assign a label $l_p(u)$ to each vertex u and p. Labeling has been used to solve many graph related problems previously [2,3,8].

Given a vertex p and the shortest path tree T_p rooted at p, let $\Pi_p(u)$ for any vertex $u \in vert(\mathcal{K})$ denote the unique tree path in T_p from p to u, and let $l_p(u)$ denote the value $m(S_i, \Pi_p(u))$. Let w denote the parent of u in tree T_p and e_{uw} denote the edge between u and w. Then $l_p(u) = l_p(w) + m(S_i, e_{uw})$. Thus for a fixed $p \in vert(\mathcal{K})$, we can traverse the tree T_p from the root to the leaves and compute the label $l_p(u)$ for all vertices u in $O(n)$ time as $m(S_i, e)$ for every edge is already precomputed earlier in the procedure UPDATE and can be queried in O(1) time. Thus the total time to compute labels $l_p(u)$ for all $p, u \in vert(\mathcal{K})$ is $O(n^2)$.

Now given a fixed vertex p and the shortest path tree T_p, we consider every cycle $C(p, e)$, where $e = (u, v)$ is a non-tree edge. We partition the cycle into three parts: the tree path $\Pi_p(u)$, the tree path $\Pi_p(v)$ and the edge e. Thus $m(S_i, C(p, e)) = m(S_i, \Pi_p(u)) + m(S_i, \Pi_p(v))) + m(S_i, e) = l_p(u) + l_p(v) + m(S_i, e)$, which can be computed in $O(1)$ time as all labels are precomputed. Note that there are $O(n^2)$ cycles in the candidate set $\mathcal{G}$ to be computed. It results that in $O(n^2)$ total time, one can compute $m(S_i, C)$ for all cycles $C \in \mathcal{G}$ and find the smallest one.

3.2 Correctness and Time Complexity

To prove the correctness of Algorithm 1, it is crucial to guarantee that the support vectors S_is and the cycles C_is satisfy the desirable properties. First, the set of support vectors $\{S_1, S_2, \cdots, S_g\}$ is a basis of $\{0, 1\}^g$ because of the construction of $\hat{S}_i$s in the procedure UPDATE. The property that $\forall j < i$, $m(S_i, C_j) = 0$ holds, because the procedure UPDATE ensures that S_i is taken as a non-trivial solution to a set of linear equations $m(x, C_j) = 0$, $1 \leq j \leq i - 1$, which always admits at least one solution. Similarly, for any $i \leq g$, there exists at least one cycle C such that the equation $m(S_i, C) = 1$ holds since both $S_1, \ldots, S_i$ and $C_1, \ldots, C_{i-1}$ at this point only form partial basis of a space with dimension g. In the base case, SHORTESTCYCLE computes this cycle C satisfying exactly this property. Then, Theorem 3.1 ensures the correctness of the algorithm.

The total running time of our algorithm is $O(n^2 g + n^\omega)$ and the analysis is as follows. The time to annotate edges and construct the candidate set is $O(n^\omega + n^2 \log n) = O(n^\omega)$ from Propositions 2.1 and 3.1. When computing the basis, the time of the procedure CYCLEBASIS is dominated by the time of EXTENDBASIS. For each $i \leq g$, the time complexity of EXTENDBASIS(i, k) is bounded by the following recurrence:

$$T(i,k) = \begin{cases} \text{the time of } \textsc{ShortestCycle}(S_i) & k = 1 \\ 2T(\cdot, k/2) + O(k^{\omega-1} n) & k > 1 \end{cases}$$

Note that in the recursion, only the second parameter k counts for the time complexity. Actually for each $i \leq g$, the time complexity of SHORTESTCYCLE(S_i) in the base case is only $O(n^2)$ as we argued earlier, that is, $T(\cdot, 1) = O(n^2)$. Then the recurrence solves to $T(\cdot, k) = O(k(n^2) + k^{\omega-1} n)$. It follows that $T(1, g) = O(n^2 g + g^{\omega-1} n)$. Combined with the time for computing annotations and constructing the candidate set, the time complexity is $O(n^2 g + n^\omega)$.

4 An Approximate Minimal Homology Basis of $\mathsf{H}_1(\mathcal{K})$

In this section, we present an algorithm to compute an approximate minimal 1-dimensional homology basis, where the approximation is defined as follows.

Definition 4.1 (Approximate minimal homology basis). *Suppose $\mathcal{C}^*$ is a minimal homology basis for $\mathsf{H}_1(\mathcal{K})$, and let $\ell_1^* \leq \ell_2^* \leq \cdots \leq \ell_g^*$ denote the sequence of sizes of cycles in $\mathcal{C}^*$ sorted in non-decreasing order. A set of g cycles $\mathcal{C}'$ is a c-*approximate minimal homology basis for $\mathsf{H}_1(\mathcal{K})$ *if (i)* $\{[C], C \in \mathcal{C}'\}$ *form a basis for* $\mathsf{H}_1(\mathcal{K})$*; and (ii) let* $\ell_1, \ldots, \ell_g$ *denote the sequence of sizes of cycles in* $\mathcal{C}'$ *in non-decreasing order, then for any* $i \in [1, g]$, $\ell_i^* \leq \ell_i \leq c \cdot \ell_i^*$.

In what follows, we provide a 2-approximation algorithm running in $O(n^\omega \sqrt{n \log n})$ time. At the high level, we first compute a candidate set $\mathcal{G}'$ of cycles that guarantees to contain a 2-approximate minimal homology basis. We then extract a 2-approximate basis from the candidate set $\mathcal{G}'$.

First, we explain the construction of a candidate set of cycles. Recall that in Sect. 3.1, we compute $O(n^2)$ candidate cycles, each of which has the form $C(p, e)$, formed by e together with the two tree-paths from root p to each of the endpoint of e within the shortest path tree T_p. We now apply the algorithm by Kavitha et al. [12] which can compute a smaller candidate set $\mathcal{G}'$ of $O(n\sqrt{n \log n})$ cycles which is guaranteed to contain a 2-approximate minimal *cycle basis* (not homology basis) for graph $\mathcal{K}^{(1)}$ (i.e., 1-skeleton of the complex $\mathcal{K}$) in $O(n\sqrt{n} \log^{3/2} n)$ *expected time*. Here, a cycle basis $\Gamma = \{\gamma_1, \ldots, \gamma_L\}$ of the graph $G = \mathcal{K}^{(1)}$ where $L = rank(\mathsf{Z}_1)$ is simply a set of cycles such that any other cycle from G can be represented uniquely as a linear combination of cycles in Γ. A *minimal cycle basis* is a cycle basis Γ^* whose total weight $\sum_{\gamma \in \Gamma^*} \mu(\gamma)$ is smallest among all cycle basis. A cycle basis Γ is a c-approximate minimal cycle basis if its total weight is at most c times that of the minimal cycle basis, i.e., at most $c \cdot \sum_{\gamma \in \Gamma^*} \mu(\gamma)$.

Now let the size $\mu(\gamma)$ of a cycle be the total weight of all edges in γ. Then, it turns out that, $\mathcal{G}'$ not only contains a 2-approximate minimal cycle basis w.r.t. this size, it also satisfies the following stronger property as proven in [12].

Proposition 4.1 ([12, Lemma 6.3])**.** *There exists a minimal cycle basis* $\Gamma^* = \{\gamma_1^*, \ldots, \gamma_L^*\}$ *such that, for any* $1 \leq i \leq L$, *there is a subset* $\Gamma_i \subseteq \mathcal{G}'$ *of the computed candidate set* $\mathcal{G}'$ *so that (i)* $\gamma_i^* = \sum_{\gamma \in \Gamma_i} \gamma$ *and (ii) each cycle in* Γ_i *has size at most* $2\mu(\gamma_i^*)$.

Next, we prove that a candidate set $\mathcal{G}'$ satisfying conditions in Proposition 4.1 is guaranteed to also contain a 2-approximate minimal homology basis. We remark that if Proposition 4.1 does not hold, then the sole condition that $\mathcal{G}'$ contains a c-approximate minimal *cycle basis* is **not** sufficient to guarantee that it also contains a c-approximate minimal *homology basis* for any constant c. A counter-example is given at the end of this section.

Lemma 4.1. *Given a set* $\mathcal{G}'$ *of cycles satisfying Proposition 4.1, there exists a minimal homology basis* $\mathcal{C}^* = \{C_1^*, \ldots, C_g^*\}$ *such that* $\mathcal{G}'$ *contains* g *cycles* $A_1, \cdots, A_g$ *with (i)* $[A_1], \cdots, [A_g]$ *form a homology basis, and (ii)* $\mu(A_i) \leq 2\mu(C_i^*)$, *for* $i = 1, \cdots, g$.

Proof. Let Γ^* be a minimal homology basis which satisfies Proposition 4.1. It is known that it contains a minimal homology basis, which we set as $\mathcal{C}^* =$

$\{C_1^*, \ldots, C_g^*\}$. Now by Proposition 4.1, for each C_i^*, there exists a subset $\Gamma_i \subseteq \mathcal{G}'$ such that $C_i^* = \sum_{\gamma \in \Gamma_i} \gamma$ and $\mu(\gamma) \leq 2\mu(C_i^*)$, $\forall \gamma \in \Gamma_i$. Assume w.l.o.g. that cycles in $\mathcal{C}^*$ are in non-decreasing order of their sizes. We now prove the lemma inductively. In particular,

Claim-A: For any k, we show that there exists $A_1, \ldots, A_k \in \bigcup_{r \leq k} \Gamma_r$ such that for each $i \in [1, k]$, (Cond-1) $\mu(A_i) \leq 2\mu(C_i^*)$; and (Cond-2) $[A_1], \ldots, [A_k]$ are independent.

The base case is straightforward: We can simply take A_1 as any cycle from Γ_1 that is not null-homologous (which must exist as $C_1^* = \sum_{\gamma \in \Gamma_1} \gamma$ is not null-homologous).

Now suppose the claim holds for k. Consider the case for $k+1$. By induction hypothesis, there exists $A_1, \ldots A_k \in \bigcup_{r \leq k} \Gamma_r$ such that (Cond-1) and (Cond-2) hold. Now consider cycles in $\bigcup_{r \leq k+1} \Gamma_r$. Let $\mathcal{H}_{k+1}$ denote the subgroup of $\mathsf{H}_1(\mathcal{K})$ generated by the homology classes of all cycles in $\bigcup_{r \leq k+1} \Gamma_r$. Note that $\mathcal{H}_{k+1}$ spans $\{[C_1^*], \ldots, [C_{k+1}^*]\}$, then the rank of $\mathcal{H}_{k+1}$ is at least $k+1$, which means there always exists a cycle $A_{k+1} \in \bigcup_{r \leq k+1} \Gamma_r$ such that $[A_{k+1}]$ is independent of $[A_1], \ldots [A_k]$. By definition of $\bigcup_{r \leq k+1} \Gamma_r$, there is an index $j \leq k+1$ such that $\mu(A_{k+1}) \leq \mu(C_j^*) \leq \mu(C_{k+1}^*)$ which satisfies both (Cond-1) and (Cond-2). Thus Claim-A holds for $k+1$ as well.

The lemma then follows when $k = g$. □

So far we have proved that the new candidate set $\mathcal{G}'$ always contains a 2-approximate minimal homology basis. What remains is to describe how to compute such an approximate basis from the candidate set $\mathcal{G}'$. First, we compute the annotation of all edges in $O(n^\omega)$ time. Let $a(e)$ denote the annotation of an edge $e \in \mathcal{K}^{(1)}$ in the complex $\mathcal{K}$; recall that $a(e)$ is a g-bit vector with $g = rank(\mathsf{H}_1(\mathcal{K}))$. Also recall that given a cycle γ, its annotation $a(\gamma) = \sum_{e \in \gamma} a(e)$ represents the homology class of this cycle, and two cycles are homologous if and only if they have the same annotation vectors.

Now order the cycles in $\mathcal{G}' = \{\gamma_1, \ldots, \gamma_m\}$, where $m = |\mathcal{G}'| = O(n\sqrt{n \log n})$, in non-decreasing order of their sizes. We will compute the annotation of all cycles in $\mathcal{G}'$ and put them in the $g \times m$ matrix M, whose i-th column $M[i]$ represents the annotation vector for the cycle γ_i. Since $\mathcal{G}'$ contains a homology basis of $\mathsf{H}_1(\mathcal{K})$ (Lemma 4.1), $rank(M) = g$.

First, we explain how to compute annotation matrix M efficiently. Let $edge(\mathcal{K}) = \{e_1, \ldots, e_L\}$ denote all edges from $\mathcal{K}$. Let A denote the $L \times m$ matrix where $\gamma_i = \sum_{j \in [1,L]} A[i][j] e_j$; that is, non-zero entries of the i-th column $A[i]$ encode all edges in the cycle γ_i. Let B denote the $g \times L$ matrix where the i-th column $B[i]$ encodes the annotation of edge e_i. It is easy to see that $M = A^T \cdot B^T$. Instead of computing the multiplication directly, we partition the matrix A^T top-down into m/L submatrices each of size at most $L \times L$. For each of this submatrix, its multiplication with B^T can be done in $O(L^\omega)$ matrix multiplication time. Thus the total time to compute the multiplication $M = A^T \cdot B^T$ takes $O(\frac{m}{L} L^\omega) = O(mn^{\omega-1})$ time as $L \leq n$. In other words, we can compute the annotation matrix M in $O(n^\omega \sqrt{n \log n})$ as $m = O(n\sqrt{n \log n})$.

We now compute a 2-approximate minimal homology basis from $\mathcal{G}'$. Here we use so-called *earliest basis*. Specifically, in general, given a matrix D with rank r, the set of column vectors $\{D[i_1], \cdots, D[i_r]\}$ is called an **earliest basis** for the vector space spanned by all columns in D (or simply, for D), if the column indices $\{i_1, \ldots, i_r\}$ are the lexicographically smallest index set such that the corresponding columns of D have full rank.

Proposition 4.2 ([3]). *Let D be an $m \times n$ matrix of rank r with entries over $\mathbb{Z}_2$ where $n \leq m$, then there is an $O(mn^{\omega-1})$ time algorithm to compute the earliest basis of D.*

Let $\{i_1, \ldots, i_g\}$ be the indices of columns in the earliest basis of M. This can be done in $O(mg^{\omega-1}) = O(n\sqrt{n \log n} \cdot g^{\omega-1})$ time by the above proposition as $m = O(n\sqrt{n \log n})$. The cycles corresponding to these columns form a homology basis by the properties of annotations [3].

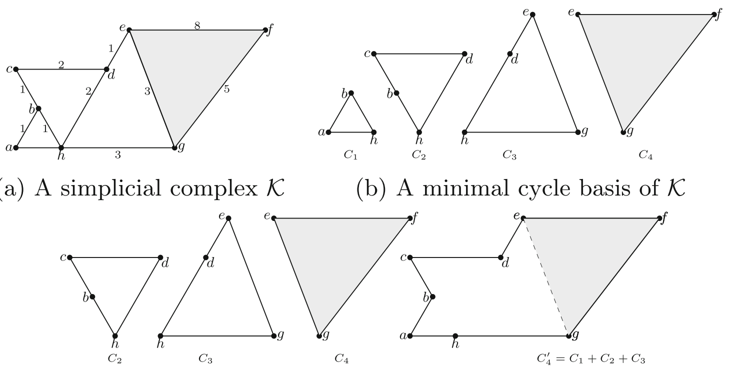
Finally, we note that the earliest basis of M has the smallest (lexicographically) sequence of size sequence. Hence its total size is at most the size of the 2-approximate minimal homology basis $A_1, \ldots, A_g$ as specified in Lemma 4.1. Hence putting everything together, we conclude with the following theorem.

Theorem 4.1. *The algorithm above computes a 2-approximate minimal homology basis of the 1-dimensional homology group $\mathsf{H}_1(\mathcal{K})$ of a simplicial complex with non-negative weights in $O(n^{\omega}\sqrt{n \log n})$ expected time.*

Remark. Since an approximate minimal homology basis still forms a basis for $\mathsf{H}_1(\mathcal{K})$, it means that computing it is at least as hard as computing the rank of $\mathsf{H}_1(\mathcal{K})$. Currently the best algorithm for the rank computation for general simplicial complex $\mathcal{K}$ is $O(n^{\omega})$ (the matrix multiplication time). Hence the best we can expect for computing an approximate minimal homology basis is perhaps $O(n^{\omega})$ (versus the $O(n^2 g + n^{\omega})$ time complexity of the exact algorithm from Sect. 3.1). We remark that we can also develop an algorithm that computes a $(2k-1)$-approximate minimal homology basis in time $O(kn^{1+1/k} g \text{ polylog } n + n^{\omega})$, where $k \geq 1$ is an integer – indeed, as the approximation factor reaches $\log n$, the time complexity becomes $O(n^{\omega})$ (which is the best time known for rank computation). The framework of this algorithm follows closely from an approach by Kavitha et al. in [12], and we thus omit the details here.

A counter-example. Figure 1 gives an example which shows that, without Proposition 4.1, it is not guaranteed that a candidate set containing a c-approximate minimal cycle basis includes a 2-approximate minimal homology basis. Let the size of a 1-cycle in $\mathcal{K}$ shown in the figure be the sum of all edges in the cycle. There is only one minimal cycle basis in this figure, namely C_1, C_2, C_3 and C_4, as shown in Fig. 1b. The minimal homology basis of $\mathcal{K}$ should be $\{C_1, C_2, C_3\}$. However, consider the candidate set $\mathcal{G}$ which contains 4 cycles as shown in Fig. 1c: C_2, C_3, C_4 and $C_4' = C_1 + C_2 + C_3$. It is easy to check that these 4 cycles in $\mathcal{G}$ form a 2-approximate minimal *cycle* basis. However, the smallest homology basis contained in $\mathcal{G}$, namely $C_2, C_3, C_4'(= C_1 + C_2 + C_3)$ is not a 2-approximate minimal homology basis.

We can make this example into a counter-example for any constant factor approximation, by adding more C_i''s (triangles) to the sequence, each of which is larger than the previous one and is also filled in. In other words, the optimal homology basis remains $\{C_1, C_2, C_3\}$, while the smallest-size homology basis from the c-approximate minimal cycle basis is $\{C_2, C_3, \sum_{i>1} C_i\}$.



(a) A simplicial complex $\mathcal{K}$ (b) A minimal cycle basis of $\mathcal{K}$

(c) A 2-approximate minimal cycle basis of $\mathcal{K}$

Fig. 1. An example where an approximate minimal cycle basis **does not** contain an approximate minimal homology basis.

5 Generalizing the Size Measure

The 1-skeleton $\mathcal{K}^{(1)}$ of the simplicial complex $\mathcal{K}$ is the set of vertices and edges in $\mathcal{K}$. If there are non-negative weights defined on edges in $\mathcal{K}^{(1)}$, it is natural to use the induced shortest path distance in $\mathcal{K}^{(1)}$ (viewed as a weighted graph) as a metric for vertices V in $\mathcal{K}$. One can then measure the "size" of a cycle to be the sum of edge weights. Indeed, this is the distance and the size measure considered in Sects. 3 and 4. In this section, we show that the algorithmic framework in Algorithm 1 can in fact be applied to a more general family of size measures. Specifically, first, we introduce what we call the *path-dominated distance* between vertices of $\mathcal{K}$ (which is not necessarily a metric). Based on such distance function, we then define a family of "size-functions" under which measure we can always compute a minimal homology basis using Algorithm 1. The shortest-path distance/size measure used in Sect. 3, and the geodesic ball-based measure proposed in [10] are both special cases of our more general concepts. We also present another natural path-dominated distance function induced by a (potentially complex) map $F : vert(\mathcal{K}) \to Z$ defined on the vertex set $vert(\mathcal{K})$ of $\mathcal{K}$ (where Z is another metric space, say $\mathbb{R}^d$). As a result, we can use Algorithm 1 to compute the shortest 1-st homology basis of $\mathcal{K}$ induced by a map $F : vert(\mathcal{K}) \to Z$.

5.1 Path-Dominated Distance

Given a connected simplicial complex $\mathcal{K}$, suppose we are given a *distance function* $d : vert(\mathcal{K}) \times vert(\mathcal{K}) \to \mathbb{R}^+ \cup \{0\}$. We now introduce the following *path-dominated distance function.*

Definition 5.1 (Path-dominated distance). *A function $d : vert(\mathcal{K}) \times vert(\mathcal{K}) \to \mathbb{R}^+ \cup \{0\}$ is a* ***path-dominated distance function (w.r.t.*** $(\mathcal{K})$***)*** *if*

(i) $d(x, y) \geq 0$ *and* $d(x, x) = 0$ *for any* $x, y \in vert(\mathcal{K})$;
(ii) given any two vertices $x, y \in vert(\mathcal{K})$*, there exists a path* π^* *connecting* x *to* y *in the 1-skeleton* $\mathcal{K}^{(1)}$ *such that* $d(x, y) = \max_{u \in vert(\pi^*)} d(x, u)$.

If edges in the 1-skeleton $\mathcal{K}^{(1)}$ have positive weights, then, it is easy to verify that the standard shortest path distance metric induced by $\mathcal{K}^{(1)}$ (viewed as a weighted graph) is path-dominated. However, note that a path-dominated distance may not be a metric. Indeed, we will shortly present a function-induced distance which is not symmetric.

We now define “shortest path” in $\mathcal{K}^{(1)}$ induced by a path-dominated distance function.

Definition 5.2 (Path-dominated shortest path). *Given any* $x, y \in vert(\mathcal{K})$*, a path* $\pi^* = \langle u_0 = x, u_1, \ldots, u_k = y$ *connecting* x *to* y *via edges in* $\mathcal{K}$ *is a* ***path-dominated shortest path in*** $\mathcal{K}$ *if for each* $i \in [1, k]$, $d(x, u_i) = \max_{j \leq i} d(x, u_j)$.

Note that this implies that any prefix of a path-dominated shortest path is also a path-dominated shortest path. The proof of the following statement is reasonably simple and can be found in Appendix B.

Claim 5.1. *A path-dominated shortest path always exists for any two vertices* $x, y \in vert(\mathcal{K})$.

Function-induced distance. Very often, the domain $\mathcal{K}$ may come with additional data modeled by a function $F : vert(\mathcal{K}) \to Z$ defined on vertices of $\mathcal{K}$, where the co-domain (Z, d_Z) is a metric space. For example, imagine that $\mathcal{K}$ represents the triangulation of a region on earth, and at each vertex, we have collected d sensor measurements (e.g. temperature, wind speed, sun-light strength, etc.), which can be modeled by a function $F : vert(\mathcal{K}) \to \mathbb{R}^d$. It is then natural to define a distance, as well as a size measure later, that depends on this function F. We introduce the following *function-induced distance* $d_F : vert(\mathcal{K}) \times vert(\mathcal{K}) \to \mathbb{R}$:

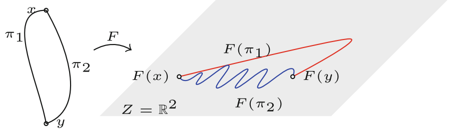


Fig. 2. The left is the original simplicial complex. There are two paths, π_1 and π_2, connecting vertices x and y. The right figure is their image under the map F with $Z = \mathbb{R}^2$ (i.e., $d_Z(\cdot, \cdot) = \| \cdot - \cdot \|$). The path π_2 is a path-dominated shortest path from x to y.

Definition 5.3. *Given any function $F : vert(\mathcal{K}) \rightarrow Z$, we define the F-**induced distance** $d_F(x, y)$ as follows:*

$$d_F(x,y) = \min_{path\ \pi(x,y) \subseteq \mathcal{K}^{(1)}} \max_{u \in \pi(x,y)} d_Z(F(x), F(u)), \tag{1}$$

where the minimum ranges over all path $\pi(x, y)$ from $K^{(1)}$ connecting x to y.

Intuitively, given a path π from x to y, $\max_{u \in \pi} d_Z(F(x), F(u))$ measures the maximum distance in terms of the function value F between the starting point x to any point in the path π, i.e., the maximum function distortion from x to π. $d_F(x, y)$ is the smallest function distortion (w.r.t. x) needed to connect from x to y. For example, in Fig. 2, the path π_2 is a path-dominated shortest path from x to y, as its image $F(\pi_2)$ has a smaller maximum distance (in terms of $d_Z = \|\cdot\|$) than the image of $F(\pi_1)$. By the definition of function-induced distance, we have:

Claim 5.2. *Given $F : vert(\mathcal{K}) \rightarrow Z$, the F-induced distance d_F is path-dominated.*

5.2 Size-Measure for 1-Cycles

Previously, the most popular way to measure the "size" of a 1-cycle is the sum of weights of edges in the cycle. Another natural measure formulated by Chen and Freedman [10] uses the minimum radius of any metric ball (centered at some vertex in $\mathcal{K}$) containing a cycle as its size. Intuitively, given a homology class, a smallest cycle of this class under this radius-measure corresponds to a cycle which is most "localized" (contained within a smallest possible metric ball). Using the shortest-path metric induced by weights on edges in $\mathcal{K}$, Chen and Freedman showed that a minimal homology basis under this radius-measure can be computed in polynomial time for any fixed-dimensional homology group. In what follows, we introduce a family size measures, which we refer to as *tight-size functions*, which generalize the radius-measure of Chen and Freedman as well as the general sum-of-weights measure. We then show that the algorithm from Sect. 3 can be used to compute a minimal homology basis for $\mathsf{H}_1(\mathcal{K})$ w.r.t. such tight-size functions.

We use the concept of *edge-short* cycles introduced in e.g. [13], whose origin traces back to [7].

Definition 5.4. *A 1-cycle C in a complex $\mathcal{K}$ is called **edge-short**, if $\mathcal{K}$ contains a vertex w, an edge $e = (u, v)$, a shortest path from w to u and a shortest path from w to v such that C is the edge disjoint union of e and the two paths.*

In our case, instead of using the shortest path metric induced by weights on the 1-skeleton $\mathcal{K}^{(1)}$ of $\mathcal{K}$, we use any path-dominated distance function d, and the "shortest paths" in the above definition will be replaced by path-dominated shortest paths in $\mathcal{K}$ w.r.t. d. To emphasize the dependency on the path-dominated distance function d, we say that a cycle C is *edge-short w.r.t. d* if the condition in Definition 5.4 holds w.r.t. path-dominated shortest paths w.r.t. d.

Definition 5.5 (Tight-size function). *Suppose $vert(\mathcal{K})$ is equipped with a path-dominated distance function d. Let $\mathsf{Z}_1(\mathcal{K})$ represent the 1-dimensional cycle group of $\mathcal{K}$. A function $\mu : \mathsf{Z}_1(\mathcal{K}) \to \mathbb{R}$ is a* ***tight-size function (w.r.t.*** *d) if under this function, there exists a minimal homology basis for $\mathsf{H}_1(\mathcal{K})$ in which all cycles are edge-short w.r.t. the path-dominated distance d.*

We may omit the reference to the path-dominated distance d when its choice is fixed or clear.

We now prove that if a function is a tight-size function, the Algorithm 1 can be used to compute a minimal homology basis. First, observe the following, which is implied by Theorem 3.1.

Claim 5.3. *If the candidate set $\mathcal{G}$ contains a minimal homology basis, then the framework Algorithm 1 will compute a minimal homology basis from the candidate set.*

What remains is to show how to compute a candidate set containing a minimal homology basis. For simplicity, from now we fix a path-dominated distance function d, and simply use *shortest paths* to refer to the *path-dominated shortest paths w.r.t. d.* We assume that the shortest paths are unique – In Appendix C, we describe how to guarantee this uniqueness condition (by assigning certain order to the shortest paths), and show that the shortest path tree T_p encoding all unique path-dominated shortest paths to any root $p \in vert(\mathcal{K})$ can be computed in $O(n \log n)$ time (with $n = |\mathcal{K}^{(1)}|$) by the standard approach.

We now construct a candidate set $\mathcal{G}$ in the same manner as in Sect. 3. First for every vertex p, we build a candidate set $\mathcal{G}_p$. Let $\Pi_p(u, v)$ denote the unique tree path between two vertices u and v. For every nontree edge $e = (u, v)$, $C(p, e) = e \circ \Pi_p(u, v)$ is a cycle. We add all such $C(p, e)$ into $\mathcal{G}_p$, i.e. $\mathcal{G}_p = \cup_{e \in E \setminus edge(T_p)} C(p, e)$. Taking the union of all such candidate sets, $\mathcal{G}$ can be constructed as $\mathcal{G} = \cup_{p \in vert(\mathcal{K})} \mathcal{G}_p$.

Lemma 5.1. *The candidate set $\mathcal{G}$ contains a minimal homology basis when the size of a cycle is measured by a tight-size function w.r.t. some path-dominated distance function d.*

Proof. By results from Appendix C, we can assume that there is only a unique path-dominated shortest path between any two vertices $u, v \in vert(\mathcal{K})$, which we denote as $SP(u, v)$. Now take any edge-short cycle C. As it is edge-short, we can find a vertex w and an edge $e = (u, v)$ such that the cycle C is the disjoint union of $SP(w, u)$, $SP(w, v)$ and e. On the other hand, the unique shortest paths $SP(w, u)$ and $SP(w, v)$ are in the shortest path tree T_w. This means that $e \notin T_w$. Hence the cycle C is a candidate cycle $C(w, e)$ from the set $\mathcal{G}_w$. It then follows that the collection $\mathcal{G}$ contains all edge-short cycles. The lemma then follows from the definition of tight-size functions and Claim 5.3. $\square$

Now that we have a candidate set that contains a minimal homology basis, we can apply the divide and conquer algorithm (Algorithm 1) from Sect. 3, and by Claim 5.3, this will output a minimal homology basis. We conclude with the following main result.

Theorem 5.1. *Suppose sizes of 1-cycles are measured by a tight-size function w.r.t. a path-dominated distance function d. Then, we can compute a minimal homology basis for $\mathsf{H}_1(\mathcal{K})$ in $O(n^\omega + n^2 g)$ time, where n is the size of 2-skeleton of $\mathcal{K}$ and g is rank$(\mathsf{H}_1(\mathcal{K}))$.*

5.3 Examples of Tight-Size Functions

Sum-of-weights size function. As mentioned earlier in Sect. 5.1, given a weight function $w : edge(\mathcal{K}) \to \mathbb{R}^+$, the shortest path distance $d_{\mathcal{K}}$ induced by the 1-skeleton $\mathcal{K}^{(1)}$ (viewed as a weighted graph) is a path-dominated function. Now given weights $w : edge(\mathcal{K}) \to \mathbb{R}^+$, the size measure $\mu_w : \mathsf{Z}_1(\mathcal{K}) \to \mathbb{R}^+$ assigning $\mu_w(C) = \sum_{e \in C} w(e)$ is a tight-size function w.r.t. the shortest path distance function $d_{\mathcal{K}}$. Hence we can obtain the main result of Sect. 3 by applying Theorem 5.1 to the tight-size function μ_w.

Radius-size function. Alternatively, we now consider the radius-based size function used e.g. in [10, 14, 15]. More specifically, suppose we are given a simplicial complex $\mathcal{K}$, and a path-dominated distance function d (which may not be a metric) on $vert(\mathcal{K})$. Define the ball B_p^r centered at p of radius r to be $B_p^r = \{\sigma \in \mathcal{K} : \forall x \in vert(\sigma), d(p,x) \le r\}$. We can then define *radius-size function* $\mu_R : \mathsf{Z}_1(\mathcal{K}) \to \mathbb{R}^+$ such that $\mu_R(C)$ of a 1-cycle C is the smallest r such that $C \subseteq B_p^r$ for some $p \in vert(\mathcal{K})$.

Proposition 5.1. *μ_R is a tight-size function w.r.t. any path-dominated distance function d.*

Proof. We need to prove that there exists a minimal homology basis where each cycle inside is edge-short. Assume this is not the case. Then given any minimal homology basis $\mathcal{B}$, there exists a cycle C which is not edge-short. Suppose cycles $\mathcal{B} = \{C_1, \ldots, C_g\}$ are sorted in nondecreasing order of their radius-size, and C_i is the first cycle in $\mathcal{B}$ that is not edge-short. Let B_p^r be the smallest ball containing C_i with $p \in vert(\mathcal{K})$; that is, $\mu_R(C_i) = r$. Let T_p denote the shortest path tree rooted at p, and Q denote the set of edges in C_i which are not in T_p. Note that Q cannot be empty; otherwise, C_i cannot be a cycle as all edges in it are tree edges. For every edge $e = (u,v)$ in Q, we can construct a cycle $C(p,e)$ as $SP(p,u) + SP(p,v) + e$, where $SP(x,y)$ denote the tree path in T_p from x to y. It is easy to see that for each such cycle $C(p,e)$ with $e \in Q$, its radius-size $\mu_R(C(p,e)) \le r$ as it is completely contained within B_p^r. Note that C_i can be represented as the sum of all such $C(p,e)$, i.e. $C_i = \sum_{e \in Q} C(p,e)$. This is because that $C_i = \sum_{e \in C_i} C(p,e)$. However, for an edge $e \in C_i \cap T_p$, $C(p,e)$ is the empty set. Hence only edges from $C_i \setminus T_p (= Q)$ contribute to this sum.

Now consider the set of cycles $\mathcal{Q} = \{C(p,e) \mid e \in Q\}$. As C_i is in a minimal homology basis $\mathcal{B}$, its homology class $[C_i]$ is independent of those generated by cycles in $\mathcal{B} \setminus \{C_i\}$. Hence there exists at least a cycle $C' \in \mathcal{Q}$ such that $[C']$ is independent of the homology class of all cycles in $\mathcal{B} \setminus \{C_i\}$. Now let $\mathcal{B}' = \mathcal{B} \cup \{C'\} \setminus \{C_i\}$ which is also a homology basis. Recall that any cycle in $\mathcal{Q}$ has radius-size at most r. We have two cases: (i) If $\mu_R(C') < r (= \mu_R(C_i)$, then

$\mathcal{B}'$ has a smaller size sequence than $\mathcal{B}$, and thus $\mathcal{B}$ cannot be a minimal homology basis. Thus we have a contradiction, meaning that all cycles in $\mathcal{B}$ must be edge-short. (ii) If $\mu_R(C') = r$, then $\mathcal{B}'$ is also a minimal homology basis. If B' contains only edge-short cycles, then we are done. If not, then we identify the next cycle that is not edge-short C_j, and it is necessary that $j > i$. We then repeat the above argument with C_j. In the end, either we find a contradiction, meaning that the edge-short cycle cannot exist in the basis we are inspecting, or we manage to replace all non-edge-short cycles to be edge-short ones of equal size, and maintain a homology basis. In the latter case, we construct a minimal homology basis with only edge-short cycles. In either case, the proposition follows. □

It then follows from the above proposition that Algorithm 1 computes a minimal homology basis under the radius-size function w.r.t. any path-dominated distances in time $O(n^\omega + n^2 g)$. In particular, combining with the two path-dominated distance functions examples we have:

Example 1: $d = d_{\mathcal{K}}$, the shortest path distance induced by the weighted graph $\mathcal{K}^{(1)}$. Under this path-dominated distance, the minimal homology basis problem under the radius-size function w.r.t. $d_{\mathcal{K}}$ is exactly the 1-dimensional case of the problem studied in [10]. An $O(n^4 g)$ time algorithm was presented to solve this problem in any dimension in [10]. However, by Theorem 5.1, we can compute a minimal homology basis of in $O(n^\omega + n^2 g)$ time, which is a significant improvement when focusing on H_1 group.

Example 2: Given a function $F : \mathit{vert}(\mathcal{K}) \to Z$ defined on $\mathcal{K}$, recall that the F-induced distance d_F as introduced in Sect. 5.1 is a path-dominated function. Now set $d = d_F$. Intuitively, the radius-size function $\mu_R(C)$ w.r.t. d_F measures the radius of the smallest metric ball in the co-domain Z that contains the image $F(C)$ of the cycle C under map F. That is, $\mu_R(C)$ measures the "size" of C w.r.t. the variation in the function F. Hence we also refer to the radius-size function w.r.t. d_F as the *F-induced radius-size function.* We believe that the F-induced distance function and F-induced radius-size function are useful objects of independent interests. The minimal homology basis of $\mathcal{K}$ under such a F-induced radius-size function can also be computed in $O(n^\omega + n^2 g)$ time.

6 Conclusions

In this paper we have given improved algorithms for computing a minimal homology basis for 1-dimensional homology group of a simplicial complex. What about higher dimensional homology? For high dimensions, it is known from [16] that computing a minimum homology basis under volume measure is NP-hard. But it follows from [10] that one can extend the radius-size measure (See Sect. 5.3) to high dimensions under which an algorithm to compute a minimum homology basis in polynomial time exists. It runs in time $O(gn^4)$ where g is the rank of d-dimensional homology group H_d. We can improve this algorithm, using persistence algorithm [17] as well as annotations for d-simplices [3], so that the time complexity improves to $O(n^{\omega+1})$ which is better when $g = \Theta(n)$. The details are presented in the Appendix A.

Acknowledgements. This works is partially supported by National Science Foundation (NSF) under grants CCF-1526513, 1740761 and 1733798.

A Computing a Minimal Homology Basis for $\mathsf{H}_d(\mathcal{K})$

Let $\mathcal{K}$ be a simplicial complex with n simplices and let g be the d-dimensional Betti number, i.e. $g = rank(\mathsf{H}_d(\mathcal{K}))$. The discrete geodesic distance $d_p : vert(\mathcal{K}) \to \mathbb{R}$ from a vertex p is given by $q \mapsto dist(p, q)$ where $dist(p, q)$ is the length of the shortest path from p to q. Extending this definition to general simplices, we have $\forall \sigma \in \mathcal{K}, d_p(\sigma) = max_{q \in vert(\sigma)} d_p(q)$. Then the geodesic ball B_p^r of radius r centered at p is defined as $B_p^r = \{\sigma \in \mathcal{K} : d_p(\sigma) \leq r\}$. Clearly, $B_p^r \subseteq \mathcal{K}$, and it is a subcomplex of $\mathcal{K}$. This is because for all faces σ' of σ, $d_p(\sigma') \leq d_p(\sigma)$, which implies that all faces of a simplex in B_p^r are also in B_p^r.

The size of a cycle C is defined as $\mu(C) = min\{r : \exists p \in vert(K), s.t.\ C \subset B_p^r\}$ [18]. In words, it is the radius of the smallest ball centered at some vertex p of C, which contains C. The definition of a minimal homology basis becomes:

Definition A.1. *Given a simplicial complex $\mathcal{K}$, a set of cycles $\{C_1, C_2, \cdots, C_g\}$ with $g = rank(\mathsf{H}_d(\mathcal{K}))$ is a d-dimensional minimal homology basis if (1) the homology classes $\{[C_1], [C_2], \cdots, [C_g]\}$ constitute a homology basis and (2) the sizes $\{\mu(C_1), \mu(C_2), \ldots, \mu(C_g)\}$ are lexicographically smallest among all such bases.*

A.1 Algorithm

In this section, we describe an algorithm to compute a minimal d-dimensional homology basis where $d \geq 1$. There are two steps in the algorithm: First computing a candidate set which contains a minimal homology basis and then computing a minimal homology basis from the candidate set. All computations are over $\mathbb{Z}_2$.

Computing a candidate set. We now describe how to compute a candidate set of cycles including a minimal homology basis. We apply the persistent homology algorithm to generate the candidate set $\mathcal{C}(p)$ for a vertex p with the following filtration: Simplices are sequenced in non-decreasing order of geodesic distances $d_p(\cdot)$ while placing a simplex before all its cofaces that have the same geodesic distance. We focus on the essential homology classes computed by persistent algorithm. There are g of them. For each essential homology class h, we denote its birth time as $r_p(h)$. For any vertex p, the number of candidate cycles in $\mathcal{C}(p)$ is g. Thus, the number of cycles of the candidate set $\mathcal{C}$ is $O(gn)$.

Claim A.1 *The candidate set $\mathcal{C}$ includes a minimal homology basis.*

Proof. Suppose not. Let $\mathcal{B}$ be any minimal homology basis and the elements in $\mathcal{B}$ are sorted in nondecreasing order of their sizes. Let class C_i be the first member in $\mathcal{B}$ which is not in the candidate set and let p be the vertex such that $C_i \subset B_p^{\mu(C_i)}$ where $\mu(C_i)$ is the size of the cycle C_i. First we claim that there exists a d-simplex σ such that $d_p(\sigma) = \mu(C_i)$ and σ is a creator of C_i. If not, there is another cycle C' such that $[C'] = [C_i]$ and $\mu(C') < \mu(C_i)$. Note that the cycles generated by creators in $B_p^{\mu(C_i)}$ form a homology basis of $B_p^{\mu(C_i)}$. We prove that the geodesic ball $B_p^{\mu(C_i)}$ must include a cycle $C^* \in \mathcal{C}$ such that the following two conditions hold: (1) $\mu(C^*) \leq \mu(C_i)$. (2) $\mathcal{B} \setminus \{C_i\} \cup \{C^*\}$ is a homology basis. Condition (1) holds because $\mu(C) \leq \mu(C_i)$ for every cycle C in $B_p^{\mu(C_i)}$.

For (2), observe that there exists a homology class $[C^*]$ generated by one creator that is independent of homology classes generated by $\mathcal{B} \setminus \{C_i\}$. If no such cycle exists, any homology class generated by one creator of $B_p^{\mu(C_i)}$ can be written as a linear combination of homology classes generated by $\mathcal{B} \setminus \{C_i\}$. The homology classes generated by creators form a homology basis of $B_p^{\mu(C_i)}$ and $C_i \in B_p^{\mu(C_i)}$. It means that $[C_i]$ is not independent of $\mathcal{B} \setminus \{C_i\}$, contradicting the assumption that $\mathcal{B}$ is a homology basis. Therefore, $\mathcal{B} \setminus \{C_i\} \cup \{C^*\}$ is a homology basis.

Combining condition (1) with (2), the homology basis $\mathcal{B}' = \mathcal{B} \setminus \{C_i\} \cup \{C^*\}$ is a minimal homology basis. What is more, if we sort the cycles in $\mathcal{B}'$ in nondecreasing order of sizes, then the first i cycles in $\mathcal{B}'$ are in the candidate set $\mathcal{C}$. This is because the cycle C^* is generated by a creator of $B_p^{\mu(C_i)}$, which means that $C^* \in \mathcal{C}$. Therefore, we find a minimal homology basis all of whose cycles are in the candidate set. $\square$

Computing a minimal homology basis. In this section we discuss an algorithm to find a minimal homology basis from the candidate set. We use annotation, denoted by $a(\cdot)$, to represent and distinguish each cycle. Recall that annotation of a cycle is a g-bit vector where $g = rank(\mathsf{H}_d)$, and that two cycles are homologous if and only if their annotations are equal. We first compute the annotations for all d-simplices in $\mathcal{K}$ [3] and give them a fixed order $\sigma_1, \sigma_2, \cdots, \sigma_{n_d}$ where n_d is the number of d-simplices in $\mathcal{K}$. Suppose we sort the cycles in the candidate set in nondecreasing order of their sizes as $C_1, C_2, \cdots, C_{gn_0}$ where n_0 is the number of vertices in $\mathcal{K}$. Then, every d-cycle C_i in $\mathcal{K}$ can be denoted as $C_i = \sum_{j=1}^{n_d} \gamma_{ij}\sigma_j$ where $\gamma_{ij} \in \{0, 1\}$ and $1 \leq i \leq gn_0$. Thus, we have $a(C_i) = \sum_{j=1}^{n_d} \gamma_{ij} a(\sigma_j)$, $1 \leq i \leq gn_0$. We compute the annotations $a(C_1), a(C_2), \cdots, a(C_{gn_0})$ for all cycles $C_1, C_2, \cdots, C_{gn_0}$ in the candidate set simultaneously.

Let $X = (a(C_1)^T, a(C_2)^T, \cdots, a(C_{gn_0})^T)^T$ and $Y = (a(\sigma_1)^T, a(\sigma_2)^T, \cdots, a(\sigma_{n_d})^T)^T$. The goal is to compute X that satisfies the following equation: $X = \varGamma Y$ where $\varGamma = (\gamma_{ij})_{gn_0 \times n_d}$. The computation of the matrix X takes time $O(n^\omega g)$ using the fast matrix multiplication algorithm where $\varGamma$ is a $gn_0 \times n_d$ matrix and Y is an $n_d \times g$ matrix.

Let X' be the transposed matrix of X. The problem of computing a minimal homology basis from the candidate set $\mathcal{C}$ is equivalent to computing the earliest basis of the matrix X' [3]. According to Proposition 4.2, computing the earliest basis of X' costs us $O(ng^\omega)$ time. Combining the time $O(n^{\omega+1})$ in building the candidate set $\mathcal{C}$ and the time $O(n^\omega g)$ in computing X, we conclude that the total running time is $O(n^{\omega+1})$.

Theorem A.1. *Given a simplicial complex $\mathcal{K}$ with n simplices, there is an algorithm to compute a minimal homology basis as defined in Definition A.1 in any dimension in time $O(n^{\omega+1})$.*

B Proof of Claim 5.1

We prove this claim by induction. First, fix any source node $x \in vert(\mathcal{K})$. We sort all other vertices in $vert(\mathcal{K})$ in non-decreasing order of $d(x, y)$; that is, $d(x, y_1) \leq d(x, y_2) \leq \ldots, \leq d(x, y_s)$ with $s = |vert(\mathcal{K})| - 1$. We carry out an induction based on this order. For the base case, any path in (ii) of Definition 5.1 is necessarily a path-dominated shortest path from x to y_1: Indeed, if there is any other vertex y (other than x and y_1) in such a path, it is necessary that $d(x, y) = d(x, y_1)$ as $d(x, y_1)$ has the smallest distance to x.

Now suppose there exists a path-dominated shortest path from x to y_i for $1 \leq i \leq s$. Consider y_{i+1} and assume that there is no path-dominated shortest path from x to y_{i+1}. By Definition 5.1, there exists a path $\Pi = (u_0 = x, u_1, \ldots, u_k = y_{i+1})$ such that for every vertex $u_i \in \Pi$, $d(x, u_i) \leq d(x, y_{i+1})$. As this path violates the conditions in Definition 5.2, there must exist a pair of vertices $u_j, u_l \in \Pi$ with $j < l$ such that $d(x, y_{i+1}) \geq d(x, u_j) > d(x, u_l)$. Let l be the maximal value with which such a pair (j, l) exists. It follows that we have $d(x, u_l) \leq d(x, u_{l+1}) \leq \ldots \leq d(x, u_k)$. By inductive hypothesis, we know that there is a path-dominated shortest path Π^* from x to u_l since $d(x, u_l) < d(x, y_{i+1})$. Hence, the path Π^* concatenated with the sub-path of Π from u_l to $u_k = y_{i+1}$ gives a path-dominated shortest path from x to y_{i+1}. The claim thus follows from induction.

C Ensuring Uniqueness of Shortest Paths

In Sect. 5, we require that the path-dominated shortest path (in this section, we use shortest path for short) in $\mathcal{K}$ between any two vertices is unique. Now we show how to avoid this restriction using an idea from [19] (Fig. 3).

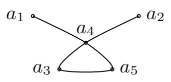


Fig. 3. Path $\pi_1 = a_1a_4a_5a_3a_4a_2$ and $\pi_2 = a_1a_4a_3a_5a_2$ are two path-dominated shortest paths from a_1 to a_2. Consider a new path $\pi = a_1a_4a_2$ which is a path-dominated shortest path from a_1 and a_2. However $len(\pi) = 2 < 5 = len(\pi_1) = len(\pi_2)$.

Lemma C.1. *Let $\mathcal{K}$ be a simplicial complex with a path-dominated distance $d(\cdot,\cdot)$. For every pair of nodes, there exists a unique shortest path π from u to v that satisfies exactly one of the following two conditions w.r.t. any other path π' from u to v:*

(1) $len(\pi) < len(\pi')$

(2) $len(\pi) = len(\pi'), min(vert(\pi) \setminus vert(\pi')) < min(vert(\pi') \setminus vert(\pi))$

Here $len(\pi)$ *denotes the number of edges in a path* π *and* $min(U)$ *denotes the minimum index of the vertices in a subset* U *of* $vert(\mathcal{K})$. *We say* $\pi < \pi'$ *if the above two conditions hold (See Fig. 3).*

The proof follows from [19, Proposition 4.1].

We now describe the algorithm to compute a shortest path tree T_p w.r.t. a path-dominated distance $d(\cdot,\cdot)$ rooted at p under the uniqueness condition. Let $\pi_p(q)$ be the tree path from p to q in the current partial tree. Let $len_p(q)$ denote the number of edges in the tree path from p to q, initially set to infinity except $len_p(p) = 0$. Initially we set a priority queue Q the vertex set $vert(\mathcal{K})$. Every time we delete a vertex q in the queue Q with the least distance $d(p, q)$, least value $len_p(q)$ and least index. We iterate for all neighbors w of q: If $\pi_p(q) \circ e$, $e = (q, w)$, is a shortest path from p to w, and is smaller than $\pi_p(w)$ as in Lemma C.1 we will update the tree path $\pi_p(w)$ in T_p as $\pi_p(q) \circ e$ and set $len_p(w) = len_p(q) + 1$. Note that those vertices not in Q will not be updated. Hence there are $O(n)$ iterations. What remains is to compute the minimum index of $vert(\pi) \setminus vert(\pi')$ given two tree paths π and π' from p to any vertex v. This can be achieved in time $O(\log n)$ adapting the algorithm from [20] for path-dominated shortest path.

Thus we conclude the above analysis with the following theorem.

Theorem C.1. *The shortest path tree in a simplicial complex* $\mathcal{K}$ *can be computed in* $O(n \log n)$ *time.*

References

1. Borradaile, G., Chambers, E.W., Fox, K., Nayyeri, A.: Minimum cycle and homology bases of surface-embedded graphs. J. Comput. Geom. **8**(2), 58–79 (2017)
2. Erickson, J., Whittlesey, K.: Greedy optimal homotopy and homology generators. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1038–1046. Society for Industrial and Applied Mathematics (2005)
3. Busaryev, O., Cabello, S., Chen, C., Dey, T.K., Wang, Y.: Annotating simplices with a homology basis and its applications. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 189–200. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31155-0_17
4. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, pp. 296–303. ACM (2014)
5. Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: A faster algorithm for minimum cycle basis of graphs. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 846–857. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27836-8_71
6. de Pina, J.C.: Applications of shortest path methods. Ph.D. thesis, University of Amsterdam (1995)
7. Horton, J.D.: A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. **16**(2), 358–366 (1987)

8. Mehlhorn, K., Michail, D.: Minimum cycle bases: faster and simpler. ACM Trans. Algorithms (TALG) **6**(1), 8 (2009)
9. Dey, T.K., Sun, J., Wang, Y.: Approximating loops in a shortest homology basis from point data. In: Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry, pp. 166–175. ACM (2010)
10. Chen, C., Freedman, D.: Measuring and computing natural generators for homology groups. Comput. Geom. **43**(2), 169–181 (2010)
11. Hatcher, A.: Algebraic Topology. Cambridge University Press, Cambridge (2002)
12. Kavitha, T., Mehlhorn, K., Michail, D.: New approximation algorithms for minimum cycle bases of graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 512–523. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70918-3_44
13. Gleiss, P.M.: Short cycles: minimum cycle bases of graphs from chemistry and biochemistry. Ph.D. thesis, Universität Wien, Austria (2001)
14. Guskov, I., Wood, Z.J.: Topological noise removal. In: 2001 Graphics Interface Proceedings, Ottawa, Canada, p. 19 (2001)
15. Wood, Z., Hoppe, H., Desbrun, M., Schröder, P.: Removing excess topology from isosurfaces. ACM Trans. Graph. (TOG) **23**(2), 190–208 (2004)
16. Chen, C., Freedman, D.: Hardness results for homology localization. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1594–1604. Society for Industrial and Applied Mathematics (2010)
17. Edelsbrunner, H., Harer, J.: Persistent homology-a survey. Contemp. Math. **453**, 257–282 (2008)
18. Chen, C., Freedman, D.: Quantifying homology classes. In: LIPIcs-Leibniz International Proceedings in Informatics, vol. 1. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2008)
19. Hartvigsen, D., Mardon, R.: The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. SIAM J. Discret. Math. **7**(3), 403–418 (1994)
20. Wulff-Nilsen, C.: Minimum cycle basis and all-pairs min cut of a planar graph in subquadratic time. arXiv preprint arXiv:0912.1208 (2009)

Shifting the Phase Transition Threshold for Random Graphs Using Degree Set Constraints

Sergey Dovgal[1,2,3(✉)] and Vlady Ravelomanana[2]

[1] LIPN – UMR CNRS 7030. Université Paris 13, 99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France
dovgal@lipn.univ-paris13.fr

[2] IRIF – UMR CNRS 8243. Université Paris 7, 8 place Aurélie Nemours, 75013 Paris, France
vlad@irif.fr

[3] Moscow Institute of Physics and Technology, Institutskiy per. 9, Dolgoprudny 141700, Russia

Abstract. We show that by restricting the degrees of the vertices of a graph to an arbitrary set $\varDelta$, the threshold point $\alpha(\varDelta)$ of the phase transition for a random graph with n vertices and $m = \alpha(\varDelta)n$ edges can be either accelerated (e.g., $\alpha(\varDelta) \approx 0.381$ for $\varDelta = \{0, 1, 4, 5\}$) or postponed (e.g., $\alpha(\{2^0, 2^1, \cdots, 2^k, \cdots\}) \approx 0.795$) compared to a classical Erdős–Rényi random graph with $\alpha(\mathbb{Z}_{\geq 0}) = \frac{1}{2}$. In particular, we prove that the probability of graph being nonplanar and the probability of having a complex component, goes from 0 to 1 as m passes $\alpha(\varDelta)n$. We investigate these probabilities and also different graph statistics inside the critical window of transition (diameter, longest path and circumference of a complex component).

1 Introduction

1.1 Shifting the Phase Transition

Consider a random Erdős–Rényi graph $G(n, m)$ [4], that is a graph chosen uniformly at random among all simple graphs built with n vertices labeled with distinct numbers from $\{1, 2, \ldots, n\}$, and m edges. The range $m = \frac{1}{2}n(1 + \mu n^{-1/3})$ where $n \to \infty$, and μ depends on n, is of particular interest since there are three distinct regimes, according to how the crucial parameter μ grows as n is large: (i) as $\mu \to -\infty$, the size of the largest component is of order $\Theta(\log n)$, and the connected components are almost surely trees and unicyclic components; (ii) next, inside what is known as the critical window $|\mu| = O(1)$, the largest component size is of order $\Theta(n^{2/3})$ and complex structures (unempty set of connected

This work is partially supported by the French project MetACOnc, ANR-15-CE40-0014 and by the French project CNRS-PICS-22479.

components having strictly more edges than vertices) start to appear with significant probabilities; (iii) finally, as $\mu \to +\infty$ with n, there is typically a unique component of size $\Theta(n)$ called the giant component. Since the article of Erdős and Rényi [4], various researchers have studied in depth the phase transition of the Erdős–Rényi random graph model culminating with the masterful work of Janson et al. [8] who used enumerative approach to analyze the fine structure of the components inside the critical window of $G(n, m)$.

The last decades have seen a growth of interest in delaying or advancing the phase transitions of random graphs. Mainly, two kinds of processes have been introduced and studied: (a) the *Achlioptas process* where models of random graph are obtained by adding edge one by one but according to a given rule which allows to choose the next edge from a set of candidate edges [1,16]; (b) the *given degree sequence models* where a sequence $(d_1, \cdots, d_n)$ of degrees is given and a simple graph built on n vertices is uniformly chosen from the set of all graphs whose degrees match with the sequence d_i (see [7,9,11,14]).

In [1,15,16], the authors studied the Achlioptas process. Bohman and Frieze [1] were able to show that there is a random graph process such that after adding $m = 0.535\,n > 0.5\,n$ edges the size of the largest component is (still) polylogarithmic in n which contrasts with the classical Erdős–Rényi random graphs. Initially, this process was conjectured to have a different local geometry of transition compared to classical Erdős–Rényi model, but Riordan and Warnke [15,16] were able to show that this is not the case. Next, in the model of random graphs with a fixed degree sequence $D = (d_1, \cdots, d_n)$, Joos et al. [9] proved that a simple condition that a graph with degree sequence D has a connected component of linear size, is that the sum of the degrees in D which are not 2 is at least $\lambda(n)$ for some function $\lambda(n)$ that goes to infinity with n. Very recently, Liebenau and Wormald [10] studied the asymptotic number of graphs with given degree sequence in a very wide range of possible degrees.

In the current work, our approach is rather different. We study *random graphs with degree constraints* that are graphs drawn uniformly at random from the set of all graphs with given number of vertices and edges with all vertices having degrees from a given set $\Delta \subseteq \mathbb{Z}_{\geq 0}$, with the only restriction $1 \in \Delta$, which we discuss below. De Panafieu and Ramos [3] calculated the asymptotic number of such graphs using the methods from analytic combinatorics. We prove that random graphs with degrees from the set Δ have their phase transition shifted from the density of edges $\frac{m}{n} = \frac{1}{2}$ to $\frac{m}{n} = \alpha$ for an *explicit* and *computable* constant $\alpha = \alpha(\Delta)$ and the new critical window of transition becomes $m = \alpha n(1 \pm \mu n^{-1/3})$.

In addition, we also prove that the structure of such graphs inside this crucial window behaves as in the Erdős–Rényi case. For instance, we prove that extremal parameters such as the diameter, the circumference or the longest path are of order $\Theta(n^{1/3})$ around $m = \alpha n$. The size of complex components of our graphs are of order $\Theta(n^{2/3})$ as μ is bounded. A very similar result but about the diameter of the largest component of $G(n, p = \frac{1}{n} + \frac{\mu}{n^{4/3}})$ has been obtained by Nachmias and Peres [12] (using very different methods).

In the seminal paper of Erdős and Rényi, they discussed the planarity of random graphs with various edge densities [4]. The probabilities of planarity inside their window of transition have been computed in [13]. In the current work, we extend this study by showing that the planarity threshold shifts from $\frac{n}{2}$ for classical random graphs to αn for graphs with degrees from Δ. More precisely, first we show that such objects are almost surely planar as μ goes to $-\infty$ and non-planar as μ tends to $+\infty$. Next, as function of μ, we compute the limiting probability that random graphs of degrees in Δ are planar as $\mu = O(1)$.

Our work is motivated by the following research questions: (i) what can be the contributions of analytic combinatorics to study constrained random graphs? (ii) the birth of the giant component often corresponds to drastic changes in the complexities of several algorithmic optimization/decision problems on random graphs, so by tuning the thresholds one can shift the location of hard random instances.

1.2 Preliminaries

The *excess* of a connected graph is the number of its edges minus the number of its vertices. For example, connected graphs with excess -1 are trees, with excess 0—graphs with one cycle (also known as unicycles or unicyclic graphs), connected bicycles have excess 2, and so on (see Fig. 1). A connected graph always has excess at least -1. A connected component with excess at least 1 is called a *complex component*. The *complex part* of a random graph is the union of its complex components.

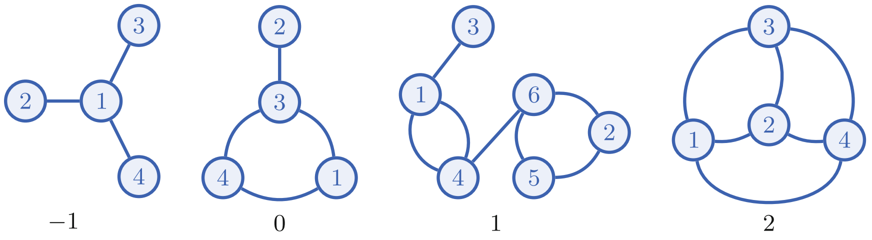


Fig. 1. Examples of connected labeled graphs with different excess. As a whole, can be considered as a graph with total excess $-1 + 0 + 1 + 2 = 2$

Next, we introduce the notion of a *2 -core (the core)* and a *3 -core (the kernel)* of a graph. The 2-core is obtained by repeatedly removing all vertices of degree 1 (smoothing). The 3-core is obtained from the 2-core by repeatedly replacing vertices of degree two and their adjacent edges by a single edge connecting the neighbors of the deleted vertex (we call this a *reduction procedure*). A 3-core can be a multigraph, i.e. there can be loops and multiple edges. There is only a finite number of connected 3-cores with a given excess [8]. Moreover, the set of 3-cores of graphs with set degree constraints (provided that $1 \in \Delta$) is the same as the set of 3-cores of classical Erdős–Rényi graphs.

The inverse images of vertices of 3-core under the reduction procedure, are called *corner vertices* (cf. Fig. 3). A *2-path* is an inverse image of an edge in a 3-core, i.e. a path connecting two corner vertices. The *circumference* of a graph is the length of its longest cycle. A *diameter* of a graph is the maximal length of the shortest path taken over all distinct pairs of vertices. It is known that the problems of finding the longest path and the circumference are NP-hard.

Random graph with degree constraints is a graph sampled uniformly at random from the set of all possible graphs $\mathcal{G}_{n,m,\Delta}$ having m edges and n vertices all of degrees from the set $\Delta = \{\delta_1, \delta_2, \ldots\} \subseteq \{0, 1, 2, \ldots\}$, see Fig. 2. The set Δ can be finite or infinite. **In this work, we require that** $1 \in \Delta$. This technical condition allows the existence of trees and tree-like structures in the random objects under consideration. We don't know what happens when $1 \notin \Delta$.

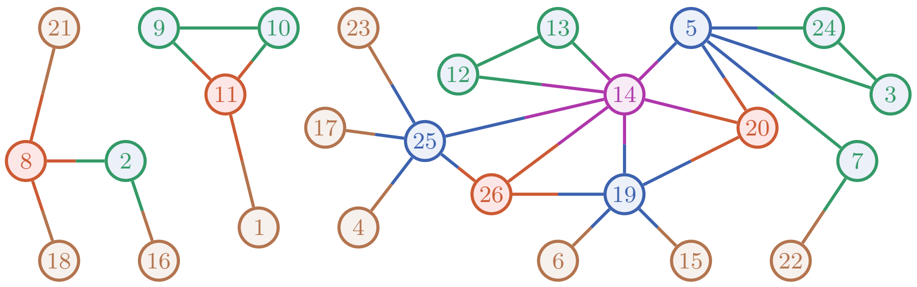


Fig. 2. Random labeled graph from $\mathcal{G}_{26,30,\Delta}$ with the set of degree constraints $\Delta = \{1, 2, 3, 5, 7\}$

The set $\mathcal{G}_{n,m,\Delta}$ is (asymptotically) nonempty if and only if the following condition is satisfied [3]:

($\mathcal{C}$) Denote $\gcd(d_1 - d_2 : d_1, d_2 \in \Delta)$ by *periodicity* p. Assume that the number m of edges grows linearly with the number n of vertices, with $2m/n$ staying in a fixed compact interval of $]\min(\Delta), \max(\Delta)[$, and p divides $2m - n \cdot \min(\Delta)$.

To a given arbitrary set $\Delta \subseteq \{0, 1, 2, \ldots\}$, we associate the *exponential generating function* (EGF) $\omega(z)$:

$$\mathrm{SET}_\Delta(z) = \omega(z) = \sum_{d \in \Delta} \frac{z^d}{d!}. \tag{1}$$

The domain of the argument z of this function can be either considered a subset $[0, R)$ of the real axis or some subset of the complex plane, depending on the context. The function $\phi_0(z) = \frac{z\omega'(z)}{\omega(z)}$, which is called the *characteristic function* of $\omega(z)$, is non-decreasing along real axis [6, Proposition IV.5], as well as the characteristic function $\phi_1(z) = \frac{z\omega''(z)}{\omega'(z)}$ of the derivative $\omega'(z)$.

The value of the threshold α, which is used in all our theorems, is a unique solution of the system of equations

$$\begin{cases} \phi_1(\widehat{z}) = 1, \\ \phi_0(\widehat{z}) = 2\alpha. \end{cases} \tag{2}$$

A unique solution $\widehat{z}$ of $\phi_1(z) = 1, z > 0$ always exists provided that $1 \in \Delta$. This solution is computable.

Structure of the Article. In Sect. 2 we state our main results and give proofs which rely on technical statements from Sect. 3. Sections 3 and 4 contain the tools from analytic combinatorics. In Sect. 5, we give the results of simulations using the recursive method from [3].

2 Phase Transition for Random Graphs

2.1 Structure of Connected Components

Recall that given a set Δ, its EGF is defined as $\omega(z) = \sum_{d\in\Delta} z^d/d!$, and characteristic function of $\omega(z)$ and its derivative $\omega'(z)$ are given by $\phi_0(z) = z\omega'(z)/\omega(z)$, $\phi_1(z) = z\omega''(z)/\omega'(z)$.

Theorem 1. *Given a set Δ with $1 \in \Delta$, let α be a unique positive solution of (2). Assume that $m = \alpha n(1 + \mu n^{-1/3})$. Suppose that Condition $(\mathcal{C})$ is satisfied and $G_{n,m,\Delta}$ is a random graph from $\mathcal{G}_{n,m,\Delta}$.*

Then, as $n \to \infty$, we have

1. *if $\mu \to -\infty$, $|\mu| = O(n^{1/12})$, then*

$$\mathbb{P}(G_{n,m,\Delta} \text{ has only trees and unicycles}) = 1 - \Theta(|\mu|^{-3}); \tag{3}$$

2. *if $|\mu| = O(1)$, i.e. μ is fixed, then*

$$\mathbb{P}(G_{n,m,\Delta} \text{ has only trees and unicycles}) \to \textit{constant} \in (0,1), \tag{4}$$
$$\mathbb{P}(G_{n,m,\Delta} \text{ has a complex part with total excess } q) \to \textit{constant} \in (0,1), \tag{5}$$

and the constants are computable functions of μ;

3. *if $\mu \to +\infty$, $|\mu| = O(n^{1/12})$, then*

$$\mathbb{P}(G_{n,m,\Delta} \text{ has only trees and unicycles}) = \Theta(e^{-\mu^3/6}\mu^{-3/4}), \tag{6}$$

$$\mathbb{P}(G_{n,m,\Delta} \text{ has a complex part with excess } q) = \Theta(e^{-\mu^3/6}\mu^{3q/2-3/4}). \tag{7}$$

Proof (Sketched). Consider a graph composed of trees, unicycles and a collection of complex connected components. Fix the total excess of complex components q. Then, there are exactly $(n - m + q)$ trees, because each tree has an excess -1.

Generating functions for each of these components are given by Lemmas 4 and 6: we enumerate all possible kernels and then enumerate graphs that reduce to them under pruning and smoothing. Let $U(z)$ be the generating function for unrooted trees, $V(z)$ be the generating function for unicycles, $E_j(z)$ be the generating functions for connected graphs with excess j. We calculate the probability for each collection $(q_1, \dots, q_k)$, while the total excess is $\sum_{j=1}^{k} j q_j = q$. Accordingly, the probability that the process generates a graph with the described property can be expressed as the ratio

$$\frac{n! \cdot |\mathcal{G}_{n,m,\Delta}|^{-1}}{(n-m+q)!} [z^n] U(z)^{n-m+q} e^{V(z)} \frac{E_1^{q_1}(z)}{q_1!} \dots \frac{E_k^{q_k}(z)}{q_k!}. \tag{8}$$

Then we use an approximation of $E_j(z)$ from Corollary 7, Lemma 6 and apply Corollary 9 with $y = \frac{1}{2} + 3q$ in order to extract the coefficients. Note that our approach is derived from the methods from [8] but due to the place limitation, most of our proofs are sketched.

2.2 Shifting the Planarity Threshold

Theorem 2. *Under the same conditions as in Theorem 1 with a number of edges $m = \alpha n(1 + \mu n^{-1/3})$, let $p(\mu)$ be the probability that $G_{n,m,\Delta}$ is planar.*

Then, as $n \to \infty$, we have uniformly for $|\mu| = O(n^{1/12})$:

1. *$p(\mu) = 1 - \Theta(|\mu|^{-3})$, as $\mu \to -\infty$;*
2. *$p(\mu) \to$ constant $\in (0, 1)$, as $|\mu| = O(1)$, and $p(\mu)$ is computable;*
3. *$p(\mu) \to 0$, as $\mu \to +\infty$.*

Proof. The graph is planar if and only if all the 3-cores (multigraphs) of connected complex components are planar. As $|\mu| = O(n^{1/12})$, Corollary 7 implies that for asymptotic purposes it is enough to consider only cubic regular kernels among all possible planar 3-cores. Let $G_1(z)$ be an EGF of connected planar cubic kernels. The function $G_1(z)$ is determined by the system of equations given in [13], and is computable. An EGF for sets of such components is given by $G(z) = e^{G_1(z)}$. We give several first terms of $G(z)$ according to [13]:

$$G(z) = \sum_{q \ge 0} g_q \frac{z^{2q}}{(2q)!^2} = 1 + \frac{5}{24} z^2 + \frac{385}{1152} z^4 + \frac{83933}{82944} z^6 + \frac{35002561}{7962624} z^8 + \dots \tag{9}$$

Thus, the number of planar cubic kernels with total excess q is given by $(2q)![z^{2q}]e^{G_1(z)} = (2q)![z^{2q}]G(z) = \frac{g_q}{(2q)!}$. In order to calculate $p(\mu)$, we sum over all possible $q \ge 0$ and multiply the probabilities that the 3-core is a planar cubic graph with excess q by the conditional probability that a random graph has planar cubic kernel of excess q.

The probability that $G_{n,m,\Delta}$ is planar on condition that the excess of the complex component is q, is equal to

$$\frac{n! |\mathcal{G}_{n,m,\Delta}|^{-1}}{(n-m+q)!} [z^n] U(z)^{n-m+q} e^{V(z)} \frac{g_q}{(2q)!} \frac{(T_3(z))^{2r}}{(1 - T_2(z))^{3r}}. \tag{10}$$

We can apply Corollary 9 and sum over all $q \geq 0$ in order to obtain the result:

$$p(\mu) \sim \sqrt{2\pi} \sum_{q \geq 0} g_q t_3^{2q} A_\Delta(3q + \tfrac{1}{2}, \mu), \tag{11}$$

where $A_\Delta(3q + \frac{1}{2}, \mu)$ and the constant t_3 are from Corollary 9. The probabilities on the borders of the transition window, i.e. $|\mu| \to \infty$ can be obtained from the properties of the function $A_\Delta(y, \mu)$.

2.3 Statistics of the Complex Component Inside the Critical Window

Theorem 3. *Under the same conditions as in Theorem 1, suppose that* $|\mu| = O(1)$, $m = \alpha n(1 + \mu n^{-1/3})$. *Then, the longest path, diameter and circumference of the complex part are of order* $\Theta(n^{1/3})$ *in probability, i.e. for each mentioned random parameter there exist computable (see Lemma 10) constants* $A, B > 0$ *depending on* Δ *such that the corresponding random variable* X_n *satisfies* $\forall \lambda > 0$

$$\mathbb{P}\left(X_n \notin n^{1/3}(A \pm B\lambda)\right) = O(\lambda^{-2}). \tag{12}$$

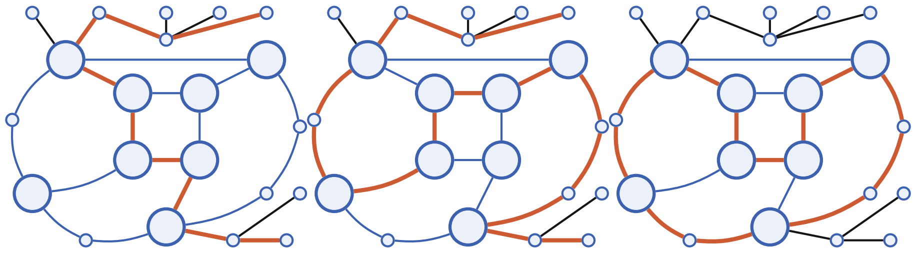


Fig. 3. Diameter, longest path and circumference of a complex component. The large vertices like ◯ are the *corner* vertices

Proof. Recall that a *2-path* is a path connecting two corner vertices inside a complex component, see Fig. 3. In Lemma 10 we prove that the length of a randomly uniformly chosen 2-path is $\Theta(n^{1/3})$ in probability. This lemma also gives the explicit expressions for A and B.

From Lemma 12 we obtain that the maximum height of sprouting tree over the complex part is also $\Theta(n^{1/3})$ in probability. Since the total excess of the complex component is bounded in probability as μ stays bounded, and the sizes of the kernels are finite, we can combine these two results to obtain the statement of the theorem, because all the three parameters come from adding/stitching several 2-paths and tree heights.

3 Saddle-Point Analysis

The crucial tool that we use in our work is the analytic lemma, Lemma 8, or equivalently, Corollary 9. Before, we develop the construction of the connected components of the graphs with set degree constraints, in the following subsection.

3.1 Symbolic Tools

For each $r \geq 0$, let us define r *-sprouted trees*: rooted trees whose vertex degrees belong to the set Δ, except the root (see Fig. 4), whose degree belongs to the set $\Delta - \ell = \{\delta \geq 0 \colon \delta + \ell \in \Delta\}$. Their EGF $T_\ell(z)$ can be defined recursively

$$T_\ell(z) = z\omega^{(\ell)}(T_1(z)),\ T_1(z) = z\omega'(T_1(z)), \quad \ell \geq 0. \tag{13}$$

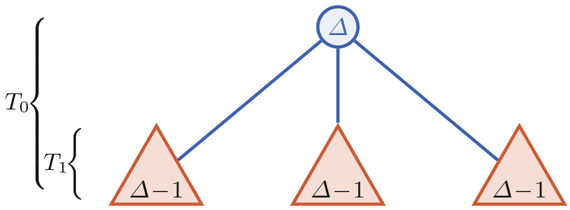


Fig. 4. Recursive construction of $T_0(z)$: the degree of the root of each subtree should belong to the set $\Delta - 1$

Lemma 4. *Let $U(z)$ be the* EGF *for unrooted trees and $V(z)$ the* EGF *of unicycles whose vertices have degrees $\in \Delta$ (Fig. 5). Then*

$$U(z) = T_0(z) - \frac{T_1(z)^2}{2}, \quad V(z) = \frac{1}{2}\left[\log\frac{1}{1-T_2(z)} - T_2(z) - \frac{T_2^2(z)}{2}\right], \tag{14}$$

where $T_0(z)$, $T_1(z)$, and $T_2(z)$ are by (13).

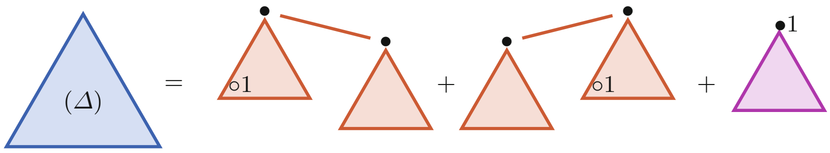


Fig. 5. Variant of dissymetry theorem for unrooted trees with degree constraints

Remark 5. Any multigraph M on n labeled vertices can be defined by a symmetric $n \times n$ matrix of nonnegative integers m_{xy}, where $m_{xy} = m_{yx}$ is the number of edges $x - y$ in M. The *compensation factor* $\kappa(M)$ is defined by

$$\kappa(M) = 1 \Bigg/ \prod_{x=1}^{n} \left(2^{m_{xx}} \prod_{y=x}^{n} m_{xy}! \right). \tag{15}$$

A *multigraph process* is a sequence of $2m$ independent random vertices

$$(v_1, v_2, \ldots, v_{2m}), \quad v_k \in \{1, 2, \ldots, n\},$$

and output multigraph with the set of vertices $\{1, 2, \ldots, n\}$ and the set of edges $\{\{v_{2i-1}, v_{2i}\} \colon 1 \le i \le m\}$. The number of sequences that lead to the same multigraph M is exactly $2^m m! \kappa(M)$ (Fig. 6).

Lemma 6. *Let $\overline{\overline{M}}$ be some 3-core multigraph with a vertex set V, $|V| = n$, having μ edges, and compensation factor $\kappa(\overline{\overline{M}})$. Let μ_{xy} be the number of edges between vertices x and y for $1 \le x \le y \le n$. The generating function for all graphs G that lead to $\overline{\overline{M}}$ under reduction is*

$$\frac{\kappa(\overline{\overline{M}}) \prod_{v \in V} T_{\deg(v)}(z)}{n!} \cdot \frac{P(\overline{\overline{M}}, T_2(z))}{(1 - T_2(z))^{\mu}}; \tag{16}$$

$$P(\overline{\overline{M}}, z) = \prod_{x=1}^{n} \left(z^{2\mu_{xx}} \prod_{y=x+1}^{n} z^{\mu_{xy}-1} (\mu_{xy} - (\mu_{xy} - 1) z) \right). \tag{17}$$

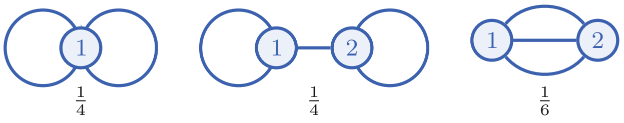


Fig. 6. All possible 3-core multigraphs of excess 1 and their compensation factors. The first one has negligible contribution because it is non-cubic

Corollary 7. *Assume that $\phi_1(\widehat{z}) = 1$. Near the singularity $z \sim \widehat{z}$, i.e. $T_2(z) \approx 1$, some of the summands from Lemma 6 are negligible. Dominant summands correspond to graphs $\overline{\overline{M}}$ with maximal number of edges, i.e. graphs with $3r$ edges and $2r$ vertices. The vertices of degree greater than 3 can be splitted into more vertices with additional edges. Due to [8, Sect. 7, Eq. (7.2)], the sum of the compensation factors is expressed as*

$$e_{r0} = \frac{(6r)!}{2^{5r} 3^{2r} (3r)! (2r)!}. \tag{18}$$

Lemma 8. *Let $m = rn = \alpha n(1 + \mu\nu)$, where $\nu = n^{-1/3}$, $|\mu| = O(n^{1/12})$, $n \to \infty$, and $\widehat{z}$ be a unique real positive solution of $\phi_1(\widehat{z}) = 1$. Let*

$$C_2 = \frac{t_3 \alpha \widehat{z}}{2(1-\alpha)}, \quad C_3 = \frac{2t_3\alpha\widehat{z}}{3}, \quad t_3 = \frac{\widehat{z}\omega'''(\widehat{z})}{\omega'(\widehat{z})}.$$

Then for any function $\tau(z)$ analytic in $|z| \leq \widehat{z}$ the contour integral encircling complex zero, admits asymptotic representation

$$\frac{1}{2\pi i}\oint (1-\phi_1(z))^{1-y} e^{nh(z;r)}\tau(z)\frac{dz}{z} \sim \nu^{2-y}(zt_3)^{1-y}\tau(z)e^{nh(z;\alpha)} \times B_\Delta(y,\mu)\Big|_{z=\widehat{z}}, \tag{19}$$

$$B_\Delta(y,\mu) = \frac{1}{3}C_3^{(y-2)/3}\sum_{k\geq 0}\frac{\left(C_2C_3^{-2/3}\mu\right)^k}{k!\Gamma\left(\frac{y+1-2k}{3}\right)} \tag{20}$$

$$h(z;r) = \log\omega' - r\log z + (1-r)\log\left(2\frac{\omega}{\omega'} - z\right)$$

Corollary 9. *If $m = \alpha n(1 + \mu n^{-1/3})$ and $y \in \mathbb{R}$, $y \geq \frac{1}{2}$, then for any $\Psi(t)$ analytic at $t = 1$ we have*

$$\frac{n!}{(n-m)!|\mathcal{G}_{n,m,\Delta}|}[z^n]\frac{U(z)^{n-m}\Psi(T_2(z))}{(1-T_2(z))^y} = \sqrt{2\pi}\Psi(1)A_\Delta(y,\mu)n^{y/3-1/6} + O(R), \tag{21}$$

$B_\Delta(y,\mu)$ is from Lemma 8 and the error term R is given by $R = (1 + |\mu|^4)n^{y/3-1/2}$. This function $A_\Delta(y,\mu)$ can be expressed in terms of $A(y,\mu) = A_{\mathbb{Z}_{\geq 0}}(y,\mu)$ introduced in [8]:

1. $A_\Delta(y,\mu) = (t_3\widehat{z})^{1-y}(3C_3)^{\frac{y-2}{3}}A\left(y, \frac{2C_2}{\sqrt[3]{(3C_3)^2}}\mu\right) = e^{-\mu^3/6}(\widehat{z}t_3)^{1-y}B_\Delta(y,\mu);$
2. *As $\mu \to -\infty$, we have* $A(y,\mu) = \dfrac{1}{\sqrt{2\pi}|\mu|^{y-1/2}}\left(1 - \dfrac{3y^2+3y-1}{6|\mu|^3} + O(\mu^{-6})\right);$
3. *As $\mu \to +\infty$, we have*

$$A(y,\mu) = \frac{e^{-\mu^3/6}}{2^{y/2}\mu^{1-y/2}}\left(\frac{1}{\Gamma(y/2)} + \frac{4\mu^{-3/2}}{3\sqrt{2}\Gamma(y/2-3/2)} + O(\mu^{-2})\right).$$

4 Method of Moments

In order to study the parameters of random structures, we apply the marking procedure introduced in [6]. We say that the variable u marks the parameter of random structure in bivariate EGF $F(z,u)$ if $n![z^nu^k]F(z,u)$ is equal to number of structures of size n and parameter equal to k. In this section we consider such parameters of a random graph as the length of 2-path, which corresponds to some edge of the 3-core, and the height of random “sprouting” tree.

4.1 Length of a Random 2-Path

Let us fix the excess vector $\mathbf{q} = (q_1, q_2, \dots, q_k)$. There are in total $q = q_1 + 2q_2 + \dots + kq_k$ connected complex components and each component has one of the finite possible number of 3-cores (see [8]). We can choose any 2-path, which is a sequence of trees, and replace it with of sequence of marked trees, see Fig. 7. Let random variable P_n be the length of this 2-path. Since an EGF for sequence of trees is $\frac{1}{1-T_2(z)}$, the corresponding moment-generating function $\mathbb{E}[u^{P_n}]$ becomes

$$\mathbb{E}[u^{P_n}] = \frac{n![z^n]\dfrac{U(z)^{n-m+q}}{(n-m+q)!}e^{V(z)}E_{\mathbf{q}}(z)\dfrac{1-T_2(z)}{1-uT_2(z)}}{n![z^n]\dfrac{U(z)^{n-m+q}}{(n-m+q)!}e^{V(z)}E_{\mathbf{q}}(z)}. \tag{22}$$

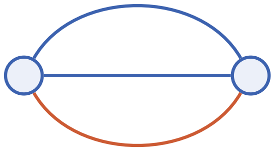


Fig. 7. Marked 2-path inside complex component of some graph

Lemma 10. *Suppose that conditions of Theorem 1 are satisfied. Suppose that there are q_j connected components of excess j for each j from 1 to k. Denote by* excess vector *a vector $\mathbf{q} = (q_1, q_2, \dots, q_k)$. Inside the critical window $m = \alpha n(1+\mu n^{-1/3})$, $|\mu| = O(1)$, the length P_n of a random (uniformly chosen) 2-path is $\Theta(n^{1/3})$ in probability, i.e.*

$$\mathbb{P}\left(P_n \notin n^{1/3} t_3 (B_1 \pm \lambda B_2)\right) \le \frac{1}{(\lambda + o(1))^2}, \quad t_3 = \widehat{z}\frac{\omega'''(\widehat{z})}{\omega'(\widehat{z})}, \tag{23}$$

$$B_1 = \frac{B_\Delta(3q+\frac{3}{2},\mu)}{B_\Delta(3q+\frac{1}{2},\mu)}, \quad B_2^2 = \frac{B_\Delta(3q+\frac{5}{2},\mu)B_\Delta(3q+\frac{1}{2},\mu) - B_\Delta^2(3q+\frac{3}{2},\mu)}{B_\Delta^2(3q+\frac{1}{2},\mu)},$$

with function $B_\Delta(y,\mu)$ from Lemma 8, $q = q_1 + 2q_2 + \dots + kq_k$.

4.2 Height of a Random Sprouting Tree

Let $\varkappa(z) = \omega'(z)$. Consider recursive definition for the generating function of simple trees whose height doesn't exceed h:

$$T^{[h+1]}(z) = z\varkappa(T^{[h]}(z)), \quad T^{[0]}(z) = 0. \tag{24}$$

The framework of multivariate generating functions allows to mark height with a separate variable u so that the function

$$F(z,u) = \sum_{n=0}^{\infty} \frac{z^n}{n!} \sum_{h=0}^{n} A_n^{[h]} u^h \tag{25}$$

is the BGF for trees, where $A_n^{[h]}$ stands for the number of simple labelled rooted trees with n vertices, whose height equals h.

Flajolet and Odlyzko [5] consider the following expressions:

$$H(z) = \frac{d}{du} F(z,u)\bigg|_{u=1}, \quad D_s(z) = \frac{d^s}{du^s} F(z,u)\bigg|_{u=1}. \tag{26}$$

Generally speaking, $H(z) = D_1(z)$ is a particular case of $D_s(z)$, but their analytic behaviour is different for $s = 1$ and $s \geq 2$.

Lemma 11 **([5, pp. 42–50]).** *The functions $H(z)$ and $D_s(z)$, $s \geq 2$ satisfy*

$$H(z) \sim \alpha \log \varepsilon(z), \quad D_s(z) \sim (\widehat{z})^{-1} s\Gamma(s)\zeta(s)\varepsilon^{-s+1}(z), \tag{27}$$

$\alpha = 2\dfrac{\varkappa'(\widehat{z})}{\varkappa''(\widehat{z})}$, $\varepsilon(z) = \widehat{z}\left(1 - \dfrac{z}{\rho}\right)^{1/2}\left(\dfrac{2\varkappa''(\widehat{z})}{\varkappa(\widehat{z})}\right)^{1/2}$, $\rho = \widehat{z}\varkappa^{-1}(\widehat{z}) = (\varkappa'(\widehat{z}))^{-1}$.

Here, $\Gamma(s)$ is a gamma-function, and $\zeta(s)$ is Riemann zeta-funciton.

It is possible to adapt this statement for the case of two kinds of sprouting trees that we have to distinguish: the first ones are attached to the vertices with degree from $\Delta - 2$, and the second — to the vertices with degree from $\Delta - 3$, we will treat these cases separately.

Lemma 12. *Inside the critical window $m = \alpha n(1 + \mu n^{-1/3})$, $|\mu| = O(1)$, the maximal height H_n of a sprouting tree, is of $O(n^{1/3})$ in probability, i.e.*

$$\mathbb{P}\left(\max H_n > \lambda n^{1/3}\right) = O(\lambda^{-2}). \tag{28}$$

5 Simulations

We considered random graphs with $n = 1000$ vertices, and various degree constraints. The random generation procedure of such graphs has been explained by de Panafieu and Ramos in [3] and for our experiments, we implemented the *recursive method.*

The generator first draws a sequence of degrees and then performs a random pairing on half-edges, as in configuration model [2]. We reject the pairing until the multigraph is simple, i.e. until there are no loops and multiple edges. As $|\mu| = O(1)$, expected number of rejections is asymptotically $\exp\left(-\frac{1}{2}\phi_1(\widehat{z}) - \frac{1}{4}\phi_1^2(\widehat{z})\right)$, which is $\exp(-3/4)$ in the critical window, and in the subcritical phase it is less.

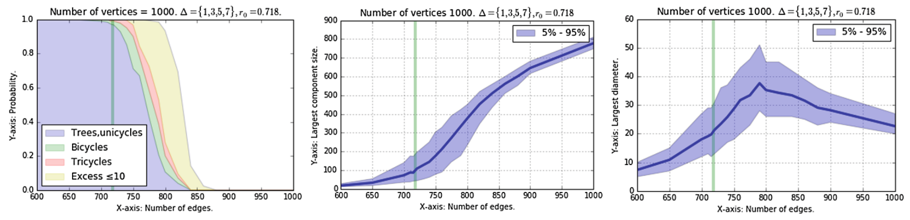
Each sequence $(d_1, \ldots, d_n)$ is drawn with weight $\prod_{v=1}^n 1/(d_v)!$. First, we use dynamic programming to precompute the sums of the weights $(S_{i,j})\colon i \in [0, n]$, $j \in [0, 2m]$ using initial conditions and the recursive expression:

$$S_{i,j} = \sum_{\substack{d_1+\cdots+d_i=j \\ d_1,\ldots,d_i \in \Delta}} \prod_{v=1}^{i} \frac{1}{d_v!}, \quad S_{i,j} = \begin{cases} 1, & (i,j) = (0,0), \\ 0, & i = 0 \text{ or } j < 0, \\ \sum\limits_{d \in \Delta} \dfrac{S_{i-1,j-d}}{d!}, & \text{otherwise.} \end{cases} \tag{29}$$

Then the sequence of degrees is generated according to the distribution

$$\mathbb{P}(d_n = d) = \frac{S_{n-1,2m-d}}{d! S_{n,2m}}. \tag{30}$$

We made plots for distributions of different parameters for $\varDelta = \{1, 3, 5, 7\}$, see Fig. 8.



(a) Largest excess (b) Largest component size (c) Graph diameter

Fig. 8. Results of experiments

Conclusion. We studied how to shift the phase transition of random graphs when the degrees of the nodes are constrained by means of analytic combinatorics [3,6]. We have shown that the planarity threshold of those constrained graphs can be shifted generalizing the results in [13]. We have also shown that when our random constrained graphs are inside their critical window of transition, the size of complex components are typically of order $n^{2/3}$ and all distances inside the complex components are of order $n^{1/3}$, thus our results about these parameters complement those of Nachmias and Peres [12].

A few open questions are left open: for given threshold value α can we find a set $\{1\} \subset \varDelta \subset \mathbb{Z}_{\geq 0}$ delivering the desired α? What happens if $1 \notin \varDelta$, for example what is the structure of random Eulerian graphs? What happens when the generating function ω itself depends on the number of vertices? Given a sequence of degrees $d_1, \ldots, d_n$ that allows the construction of a forest of an unbounded size, a first approach to study possible relationship between the models can be the computation of the generating function $\omega(z) = \sum_{i \geq 0} \mathrm{weight}(i) \dfrac{z^i}{i!}$ for a suitable weight function corresponding to $d_1, \ldots, d_n$.

Acknowledgements. We would like to thank Fedor Petrov for his help with a proof of technical condition for saddle-point analysis, Élie de Panafieu, Lutz Warnke, and several anonymous referees for their valuable remarks.

References

1. Bohman, T., Freize, A.: Avoiding a giant component. Random Struct. Algorithms **19**(1), 75–85 (2001)
2. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. Eur. J. Comb. **1**, 311–316 (1980)

3. de Panafieu, É., Ramos, L.: Enumeration of graphs with degree constraints. In: Proceedings of the Meeting on Analytic Algorithmics and Combinatorics (2016)
4. Erdős, P., Rényi, A.: On the evolution of random graphs. A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei **5**, 17–61 (1960)
5. Flajolet, P., Odlyzko, A.M.: The average height of binary trees and other simple trees. J. Comput. Syst. Sci. **25**, 171–213 (1982)
6. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge Press, Cambridge (2009)
7. Hatami, H., Molloy, M.: The scaling window for a random graph with a given degree sequence. Random Struct. Algorithms **41**(1), 99–123 (2012)
8. Janson, S., Knuth, D.E., Łuczak, T., Pittel, B.: The birth of the giant component. Random Struct. Algorithms **4**(3), 231–358 (1993)
9. Joos, F., Perarnau, G., Rautenbach, D., Reed, B.: How to determine if a random graph with a fixed degree sequence has a giant component. In: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pp. 695–703 (2016)
10. Liebenau, A., Wormald, N.: Asymptotic enumeration of graphs by degree sequence, and the degree sequence of a random graph. arXiv preprint arXiv:1702.08373 (2017)
11. Molloy, M., Reed, B.A.: A critical point for random graphs with a given degree sequence. Random Struct. Algorithms **6**(2/3), 161–180 (1995)
12. Nachmias, A., Peres, Y.: Critical random graphs: diameter and mixing time. Ann. Probab. **36**(4), 1267–1286 (2008)
13. Noy, M., Ravelomanana, V., Rué, J.: On the probability of planarity of a random graph near the critical point. Proc. Am. Math. Soc. **143**(3), 925–936 (2015)
14. Riordan, O.: The phase transition in the configuration model. Comb. Probab. Comput. **21**(1–2), 265–299 (2012)
15. Riordan, O., Warnke, L.: Achlioptas process phase transitions are continuous. Ann. Appl. Probab. **22**(4), 1450–1464 (2012)
16. Riordan, O., Warnke, L.: The phase transition in bounded-size Achlioptas processes. arXiv preprint arXiv:1704.08714 (2017)

On the Biased Partial Word Collector Problem

Philippe Duchon[1,2] and Cyril Nicaud[3](✉)

[1] University of Bordeaux, LaBRI, UMR 5800, 33400 Talence, France
[2] CNRS, LaBRI, UMR 5800, 33400 Talence, France
[3] Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, 77454 Marne-la-Vallée, France
cyril.nicaud@u-pem.fr

Abstract. In this article we consider the following question: N words of length L are generated using a biased memoryless source, i.e. each letter is taken independently according to some fixed distribution on the alphabet, and collected in a set (duplicates are removed); what are the frequencies of the letters in a typical element of this random set? We prove that the typical frequency distribution of such a word can be characterized by considering the parameter $\ell = L/\log N$. We exhibit two thresholds $\ell_0 < \ell_1$ that only depend on the source, such that if $\ell \leq \ell_0$, the distribution resembles the uniform distribution; if $\ell \geq \ell_1$ it resembles the distribution of the source; and for $\ell_0 \leq \ell \leq \ell_1$ we characterize the distribution as an interpolation of the two extremal distributions.

1 Introduction

The *coupon collector problem* is a classical topic in discrete probability; in its most basic form, the question is to determine how many independent draws from a uniform distribution on some fixed finite set E (say, of cardinality n) are needed, in expectation, to obtain each possible value at least once. The answer turns out to be exactly nH_n, where $H_n = 1 + 1/2 + \cdots + 1/n$ denotes the n-th harmonic number.

It is natural to consider non-uniform versions of the problem, where values have different probabilities. Typically, some structure is needed on the set of possible values to make the problem tractable. The *weighted word collector problem*, as studied in [1], corresponds to the case where E is a set of words of a fixed length L (possibly, all words over some finite alphabet A, *i.e.* A^L, but more generally for some language $\mathcal{L} \subseteq A^L$), each word has a probability proportional to its weight, and this weight is defined as a product of individual letter weights.

In this paper, we consider a process related to the weighted word collector process when $\mathcal{L} = A^L$. In this case, words are drawn according to a *memoryless source*: each letter a_i has a specific *a priori* probability $p(a_i)$, and words are composed of L independent letters drawn from this probability distribution. Instead of waiting for all possible words to appear, we consider a *partial word*

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 413–426, 2018.
https://doi.org/10.1007/978-3-319-77404-6_30

collector: we repeatedly draw random words from the memoryless model, keeping track of the set of distinct words "collected".
In this setting, we try to answer the following somewhat informal question:

When N independent draws have been made from the random word model, what does a "typical" member of the set of already-seen words look like?

More precisely, we study the likely *composition* of collected words; that is, the number of occurrences of each letter in a word drawn uniformly at random from the set of already collected words. At this point, it is only for convenience that we describe the process in terms of picking a random word from those already collected; our interest is in identifying what a typical collected word looks like.

This question is related to the *subword complexity* of a long random word in the memoryless model. The subword complexity function of a (finite or infinite) word w is the function that maps each integer L to the number of different factors of length L that appear in w; as shown in [4], the (random) subword complexity for factors of length L of a random word of length $N + L - 1$ in the memoryless model, is very close (in expectation, and in distribution) to the size of our partial word collection.

Our initial motivation for studying this problem follows the work of Rubinchik and Shur on the expected number of distinct palindromic factors in a uniform random word [6], which we extended to α-gapped patterns [3] (an α-gapped pattern is a factor uvu with $|uv| \leq \alpha|u|$ for given $\alpha \geq 1$). When trying to extend these results to words generated by a memoryless source, a problem very similar to the one investigated in this paper arises: typical palindromic factors have length in $\Theta(\log n)$, and a promising way to count them is to identify those who contribute the most, using (and extending) the methods presented in this paper.

Special cases of our question can be answered easily, at least informally. If all words have the same probability (*i.e.*, $p(a_i) = 1/k$ for each of the k letters), then drawing a uniform word from the set of collected words is equivalent to drawing a uniform word; by the law of large numbers, each letter is extremely likely (at least for large L) to have an observed frequency close to its *a priori* probability $1/k$.

When the letter probabilities are not uniform, the asymptotic regimes (with fixed L and variable N) are intuitively clear. For very small N, all N collected words are likely to be different, so the two-step sampling process is equivalent to drawing a single random word from the memoryless model; *i.e.* letter frequencies are likely to be close to the letter probabilities $p(a_i)$. At the other end of the spectrum, if N is large enough, with high probability all words have been collected, so that drawing a uniform word from the set of collected words is almost equivalent to drawing a uniform random word; letter frequencies should be close to the uniform $1/k$.

The interesting case lies in the intermediate regime. In the present paper, we exhibit an evolution for the likely composition of typical collected words; the precise statement is given by Theorem 1. The significant parameter is the ratio $\ell = L/\log(N)$. Informally, as N grows (as ℓ decreases), the composition goes continuously from the *a priori* composition to the *uniform* composition along a predetermined curve: at all times, each letter a_i has typical frequency proportional to $p(a_i)^c$ for some constant c that depends on the ratio ℓ; as N grows from 1 to infinity (as ℓ decreases from infinity to 0), c decreases from 1 to 0. Our main theorem also gives explicit thresholds, one until which the typical letter frequencies fit the *a priori* frequencies, and one after which they fit the uniform frequencies. Unsurprisingly, the latter is significantly smaller than the time to a full collection: typical collected words "look like" uniform words long before all words have been collected.

2 Definitions and Notations

If k is a positive integer, let $[k] = \{1, \dots, k\}$. For $\mathbf{x} = (x_1, \dots, x_k) \in \mathbb{R}^k$, let $\|\mathbf{x}\| = \sqrt{\sum_{i\in[k]} x_i^2}$ and if $d > 0$, let $\overline{B}(\mathbf{x}, d)$ be the closed ball of all vectors $\mathbf{y}$ such that $\|\mathbf{y} - \mathbf{x}\| \leq d$.

Let $A = \{a_1, \dots, a_k\}$ be an alphabet with $k \geq 2$ letters, which we fix from now on. For any word $w \in A^*$, let $\mathbf{comp}(w) = (|w|_1, \dots, |w|_k)$ denote its *composition vector*, where $|w|_i$ is the number of occurrences of a_i in w. If w is not empty, let $\mathbf{freq}(w) = (\frac{|w|_1}{|w|}, \dots, \frac{|w|_k}{|w|})$ denote its *frequency vector*.

The (natural-based) entropy function on k positive variables is defined by

$$H_k(\mathbf{x}) = H_k(x_1, \dots, x_k) = -\sum_{i\in[k]} x_i \log x_i.$$

We will omit the index k in the sequel, as it is fixed in our settings, and write $H(\mathbf{x})$ instead of $H_k(\mathbf{x})$.

Throughout the article, we assume some probability vector $\mathbf{p} = (p_1, \dots, p_k)$ to be fixed, with $p_i \neq 0$ for every $i \in [k]$, and we consider statistics in the memoryless model where each letter a_i has probability p_i. We denote by $\mathbb{P}_L$ this probability measure on A^L. We also assume that $\mathbf{p}$ is not the uniform distribution, *i.e.* there exists $i \in [k]$ such that $p_i \neq \frac{1}{k}$. Let $p_{\min} = \min_{i\in[k]} p_i$ and $p_{\max} = \max_{i\in[k]} p_i$ be the minimal and maximal values of $\mathbf{p}$.

Remark that since the number of letters $k \geq 2$ and the probability distribution $\mathbf{p}$ are fixed throughout the article, the constants we use may implicitly depend on $\mathbf{p}$ and k.

In our statements and proofs below, we mainly work on frequency vectors of words of length L, which are very specific vectors of $\mathbb{R}^k$. Depending on our needs, we will see them as frequency vectors, probability vectors (going from discrete to continuous), or even just vectors. We therefore introduce the following notations:

– For given positive integer L, let $\mathcal{F}_L$ denote the set of frequency vectors of words of length L, defined by

$$\mathcal{F}_L = \left\{ (x_1, \dots, x_k) \in \mathbb{R}^k : \sum_{i\in[k]} x_i = 1 \text{ and } \forall i \in [k],\ x_i L \in \mathbb{Z}_{>0} \right\}.$$

– Let $\mathcal{P}$ denote the set of probability vectors, defined by

$$\mathcal{P} = \left\{ (x_1, \ldots, x_k) \in \mathbb{R}^k : \sum_{i \in [k]} x_i = 1 \text{ and } \forall i \in [k],\ 0 \leq x_i \leq 1 \right\}.$$

– We will also need a restriction of $\mathcal{P}$ to probability vectors that are not close to the border. Let $\tilde{\mathcal{P}}$ be the subset of $\mathcal{P}$ defined by

$$\tilde{\mathcal{P}} = \left\{ (x_1, \ldots, x_k) \in \mathbb{R}^k : \sum_{i \in [k]} x_i = 1 \text{ and } \forall i \in [k],\ \frac{p_{\min}}{2} \leq x_i \leq 1 \right\}.$$

3 Main Result and Proof Sketch

In this section we define the problem that is studied in this paper, state our main result and provide a very informal proof sketch.

3.1 The Biased Partial Coupon Collector Problem

For any two positive integers N and L, we are interested in the following two-steps process:

1. Generate, independently, N words of length L following the memoryless distribution of parameter $\mathbf{p}$ and collect them in a set $\mathcal{S}_{N,L}$, disregarding multiplicities; if a given word is generated several times, it contributes only once to $\mathcal{S}_{N,L}$ (consequently, the number of elements in $\mathcal{S}_{N,L}$ is itself random).
2. Draw uniformly at random an element of $\mathcal{S}_{N,L}$, which we call $U_{N,L}$.

More formally, let $X_1, \ldots, X_N$ be N i.i.d. random words, each of length L and chosen using the memoryless source of parameter $\mathbf{p}$. If $\boldsymbol{u} = (u_1, \ldots, u_N)$ is a tuple of elements, of A^L, let $\text{Set}(\mathbf{u})$ be the set defined by

$$\text{Set}(\mathbf{u}) = \left\{ u \in A^L :\ \exists i \in [N],\ u_i = u \right\}.$$

The random set $\mathcal{S}_{N,L}$ is defined by $\mathcal{S}_{N,L} = \text{Set}(X_1, \ldots, X_n)$, and $U_{N,L}$ consists in choosing uniformly at random an element of $\mathcal{S}_{N,L}$ (which cannot be empty in our settings).

We mainly focus on the typical composition of letters within the result of our random process, *i.e.* we are interested in the random vector $\mathbf{freq}(U_{N,L})$.

Intuitively, our main theorem states that when (a) L is small compared to $\log N$, almost all words have been collected, hence our process is almost the same as selecting uniformly at random a word of A^L: the frequency vector resembles the uniform distribution. On the other hand, when (c) $\log N$ is small compared to L, only a few number of words have been collected, there are few duplicates, hence our process is almost the same as just generating one word with the source. The intermediate range (b) corresponds to an interpolation between the two distributions. Also, note that, seeing the process as incrementally collecting words, regime (a) occurs well before a majority of possible words have been collected, and that regime (c) lasts well after duplicates have become numerous; the precise description of the regions for the various regimes is also a part of our results.

The statement is the following.

Theorem 1. *Consider a memoryless source for a fixed alphabet of size k, of probability vector $\mathbf{p}$ that is not the uniform distribution, and define*

$$\Phi(t) = \sum_{i\in[k]} p_i^t, \ \forall t \in \mathbb{R}.$$

Let $\ell_0 = \frac{-k}{\sum_{i\in[k]} \log p_i}$ and $\ell_1 = \frac{1}{H(\mathbf{p})}$. There exist a positive integer L_0 and a positive real λ such that for every integers $L \geq L_0$ and $N \geq 2$, if we set $\ell = \frac{L}{\log N}$ then the following results hold:

(a) If $\ell \leq \ell_0$, then

$$\mathbb{P}\left(\|\mathbf{freq}(U_{N,L}) - \overline{\mathbf{x}}\| \geq \frac{\log L}{\sqrt{L}}\right) \leq L^{-\lambda \log L}, \ \textit{with } \overline{\mathbf{x}} = \left(\frac{1}{k}, \dots, \frac{1}{k}\right).$$

(b) If $\ell_0 \leq \ell \leq \ell_1$, then

$$\mathbb{P}\left(\|\mathbf{freq}(U_{N,L}) - \mathbf{x}_c\| \geq \frac{\log L}{\sqrt{L}}\right) \leq L^{-\lambda \log L}, \ \textit{with } \mathbf{x}_c = \left(\frac{p_1^c}{\Phi(c)}, \dots, \frac{p_k^c}{\Phi(c)}\right),$$

where c is the unique solution in $[0,1]$ of the equation $\ell\Phi'(c) + \Phi(c) = 0$.

(c) If $\ell \geq \ell_1$, then

$$\mathbb{P}\left(\|\mathbf{freq}(U_{N,L}) - \mathbf{p}\| \geq \frac{\log L}{\sqrt{L}}\right) \leq L^{-\lambda \log L}, \ \textit{with } \mathbf{p} = (p_1, \dots, p_k).$$

Remark 1. Observe that if $\ell = \ell_0$, then $\mathbf{x}_c = \overline{\mathbf{x}}$ and that if $\ell = \ell_1$, then $\mathbf{x}_c = \mathbf{p}$. Observe also that if we allow $\mathbf{p} = \overline{\mathbf{x}}$, then $\ell_0 = \ell_1$ and everything collapses; in this case our random process, for any value of N, is just a complicated way to produce a uniform random word of length L.

Remark 2. A reviewer suggested a change of parameterization that looks promising, by setting $N = (k^L)^\alpha$, for $\alpha \in \mathbb{R}^+$. This way the parameter is α and not ℓ, and they are related by $\alpha = \frac{1}{\ell \log k}$. Thus, as N increases for fixed L, α also increases; notice that the case $\alpha = 1$ corresponds to the situation where one draws just enough words to possibly get each existing word exactly once (though this would happen with extremely small probability). In this parameterization, the first threshold α_0, corresponding to ℓ_1, is $\alpha_0 = H_k(\mathbf{p})$, where $H_k(\mathbf{x}) = \frac{1}{\log k} H(\mathbf{x})$ is the entropy in base k of $\mathbf{x}$. The second threshold α_1, corresponding to ℓ_0, is $\alpha_1 = -\frac{1}{k \sum_i \log_k p_i}$. The reviewer also observed that α_1 can be written $\alpha_1 = 1 + \frac{1}{\log k} D_{KL}(\mathbf{u}|\mathbf{p})$, using the Kullback-Leibler divergence

[5, p. 38], which is a classical notion in information theory defined for two positive vectors $\mathbf{x}$ and $\mathbf{y}$ of size k by

$$D_{KL}(\mathbf{x}|\mathbf{y}) = \sum_{i\in[k]} x_i \log \frac{x_i}{y_i}.$$

Here the 1 term in the expression for α_1 comes from the (base k) entropy of the uniform distribution vector $\mathbf{u}$. In this form, it is more readily apparent that $\alpha_0 < 1 < \alpha_1$ holds as soon as $\mathbf{p}$ is not the uniform vector, since the Kullback-Leibler divergence is positive.

3.2 Main Steps of the Proof

For this proof sketch, the reader must be aware that we do not mention some technical conditions that are necessary for some statements to be correct. Our aim here is only to present an informal guide to the technical sections where Theorem 1 is proved.

It is convenient to introduce the parameter ℓ defined by $L = \ell \log N$, as the main phase transitions appear when the number of words is exponential in their lengths[1]. The proof consists in identifying the frequency vectors corresponding to compositions of words that contribute the more to $\mathcal{S}_{N,L}$.

For a given $\mathbf{x} \in \mathcal{F}_L$, we first prove that there are roughly[2] $N^{\ell H(\mathbf{x})}$ words of length L whose frequency vector is $\mathbf{x}$. Let $\mathcal{W}_L(\mathbf{x})$ denote this set of words.

Observe that words that share the same frequency vector $\mathbf{x}$ have the same probability $p_L(\mathbf{x}) = N^{\ell \sum_{i\in[k]} x_i \log p_i}$ of being generated by the memoryless source. Hence, the probability that a word of $\mathcal{W}_L(\mathbf{x})$ appears in $\mathcal{S}_{N,L}$ is exactly $q_{N,L}(\mathbf{x}) := 1 - (1 - p_L(\mathbf{x}))^N$. There are two cases: If $1 + \ell \sum_{i\in[k]} x_i \log p_i < 0$ then $q_{N,L}(\mathbf{x}) \approx N^{1+\ell \sum_{i\in[k]} x_i \log p_i}$; otherwise $q_{N,L}(\mathbf{x}) \approx \Theta(1)$. For $w \in \mathcal{W}_L(\mathbf{x})$, this can be summarized as follows:

$$\mathbb{P}(w \in \mathcal{S}_{N,L}) = q_{N,L}(\mathbf{x}) \approx N^{\min\left(0, 1+\ell \sum_{i\in[k]} x_i \log p_i\right)}.$$

By linearity of expectation, we can thus estimate the expected contribution to $\mathcal{S}_{N,L}$ of $\mathcal{W}_L(\mathbf{x})$ for a given frequency vector $\mathbf{x}$:

$$\mathbb{E}\,|\mathcal{S}_{N,L} \cap \mathcal{W}_L(\mathbf{x})| \approx N^{\ell \min(H(\mathbf{x}), K_\ell(\mathbf{x}))}, \text{ with } K_\ell(\mathbf{x}) = H(\mathbf{x}) + \frac{1}{\ell} + \sum_{i\in[k]} x_i \log p_i.$$

To find the frequency vectors that contribute the most, we have to study the function $G_\ell(\mathbf{x}) = \min(H(\mathbf{x}), K_\ell(\mathbf{x}))$. This is the minimum of two strictly concave functions on $\mathcal{P}$, each of which has a maximum. There are two main situations for the location of the maximum of such a function, as depicted in Fig. 1 (for functions of a single variable).

[1] It is a more natural view of the process to consider L as fixed and N as varying; this, however, leads to the somewhat artificial parameterization $N = \exp(L/\ell)$.

[2] By "roughly" we mean up to some multiplicative power of L, with $L = \Theta(\log N)$ at our scale.

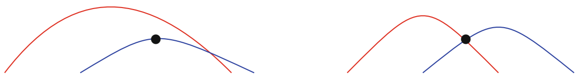


Fig. 1. The two possibilities for the location of the maximum value of g defined as the minimum of two concave functions g_1 and g_2 that both have a maximum. On the left, the situation where the maximum of g_1 (resp. g_2) is reached for some x_0 with $g_2(x_0) \geq g_1(x_0)$ (resp. $g_1(x_0) \geq g_2(x_0)$); in this case, the maximum of $g = \min(g_1, g_2)$ is $g(x_0) = g_1(x_0)$. On the right, the case were the maxima of both g_1 and g_2 do not satisfy the previous condition; the maximum of g is then located at the intersection of both curves. Note that when dealing with concave functions of several variables, this intersection is not reduced to a single point, but it still contains the maximum.

It is well known [5] that $H(\mathbf{x})$, the classical entropy function on $\mathcal{P}$, has its maximum for the uniform distribution $\overline{\mathbf{x}} = (\frac{1}{k}, \ldots, \frac{1}{k})$, with $H(\overline{\mathbf{x}}) = \log k$. Moreover, by Gibbs' inequality [5], $K_\ell(\mathbf{x})$ reaches its maximum on $\mathcal{P}$ at $\mathbf{p}$. Hence $\overline{\mathbf{x}}$ and $\mathbf{p}$ are two candidates for the first case of Fig. 1, corresponding to cases (a) and (c) of Theorem 1, respectively. Case (b) corresponds to the second case of Fig. 1, when $K_\ell(\overline{\mathbf{x}}) < H(\overline{\mathbf{x}})$ and $H(\mathbf{p}) < K_\ell(\mathbf{p})$: the maximum of $G_\ell(\mathbf{x})$ is reached for a value $\mathbf{x}$ such that $H(\mathbf{x}) = K_\ell(\mathbf{x})$, *i.e.* on the hyperplane of equation $\sum_{i\in[k]} x_i \log p_i = -\frac{1}{\ell}$. Standard techniques for multivariate differentiable functions yield that the maximum is located at $\mathbf{x}_c$ given in Theorem 1.

To turn these informal steps into a full proof of our main statement, we also need to prove that $U_{N,L}$ resemble the frequency vector that contributes the most in expectation, to $|\mathcal{S}_{N,L} \cap \mathcal{W}_L(\mathbf{x})|$, *i.e.* that the distribution of $\mathbf{freq}(U_{N,L})$ is concentrated around $\mathbf{x}_c$.

4 Proof of Theorem 1

4.1 Preliminary Results

The following lemma establishes some simple bounds for the cardinality of $\mathcal{W}_L(\mathbf{x})$, justifying the rough estimate of $N^{\ell H(\mathbf{x})}$ discussed in Sect. 3.2.

Lemma 1. *There exists a positive real constant α such that for all $\mathbf{x} \in \mathcal{P}$,*

$$|\mathcal{W}_L(\mathbf{x})| \leq \alpha e^{L\,H(\mathbf{x})} = \alpha N^{\ell\,H(\mathbf{x})}. \tag{1}$$

There exists a positive real constant β such that for all $\mathbf{x} \in \tilde{\mathcal{P}}$, such that for all $\mathbf{y} \in \overline{B}(\mathbf{x}, \frac{k}{L}) \cap \mathcal{F}_L$ we have

$$|\mathcal{W}_L(\mathbf{y})| \geq \beta L^{-\frac{k-1}{2}} e^{L\,H(\mathbf{x})} = \beta L^{\frac{k-1}{2}} N^{\ell\,H(\mathbf{x})}. \tag{2}$$

Moreover $\overline{B}(\mathbf{x}, \frac{k}{L}) \cap \mathcal{F}_L$ is not empty.

Lemma 2 below will be used to prove that sufficiently many words of $\mathcal{S}_{N,L}$ have a (well chosen) frequency vector $\mathbf{y}$. It is proved using the study of negatively associated random variables by Dubhashi and Ranjan [2].

Lemma 2. *Let $\mathbf{y} \in \mathcal{F}_L$ such that $q_{N,L}(\mathbf{y}) \geq \gamma$, for some $\gamma > 0$. Then the following inequality holds:*

$$\mathbb{P}\left(|\mathcal{W}_L(\mathbf{y}) \cap \mathcal{S}_{N,L}| \leq \frac{\gamma}{2}|\mathcal{W}_L(\mathbf{y})|\right) \leq \exp\left(-\frac{\gamma^2|\mathcal{W}_L(\mathbf{y})|}{2}\right).$$

For $\mathbf{x} \in \mathcal{P}$, let $\mathcal{B}_L(\mathbf{x})$ be the set of probability vectors that are far from $\mathbf{x}$:

$$\mathcal{B}_L(\mathbf{x}) = \left\{\mathbf{y} \in \mathcal{P} : \ \|\mathbf{y} - \mathbf{x}\| \geq \frac{\log L}{\sqrt{L}}\right\}.$$

Proposition 1 is our main tool for proving Theorem 1. The technical conditions can be seen as follows. Condition (1) is used to obtain an upper bound on the number of elements of $\mathcal{S}_{N,L}$ whose frequency vectors are in $\mathcal{B}_L$, which holds with very high probability (using the Markov inequality from the bound on the expectation). Condition (2) ensures that with high probability we have a lot of elements of $\mathcal{S}_{N,L}$ whose frequency vectors are not in $\mathcal{B}_L$; the precise statement of the condition is chosen to fit the formula within the exponential in Lemma 2.

Proposition 1. *Let ℓ^- and ℓ^+ in $\mathbb{R} \cup \{-\infty, +\infty\}$ such that $\ell^- < \ell^+$. Let L_0 be a sufficiently large integer, and $\mathbf{x} \in \tilde{\mathcal{P}}$. Assume that there exist two positive constants λ_1 and λ_2, such that, for any $L \geq L_0$ and $N \geq 2$ for which $\ell = L/\log(N)$ satisfies $\ell^- \leq \ell \leq \ell^+$; then for any $\mathbf{y} \in \overline{B}(\mathbf{x}, k/L)$ the following two conditions hold:*

(1) $\mathbb{E}|\mathcal{W}_L(\mathcal{B}_L(\mathbf{x})) \cap \mathcal{S}_{N,L}| \leq L^{-\lambda_1 \log L} N^{\ell H(\mathbf{x})} q_{N,L}(\mathbf{y})$;
(2) $N^{\ell H(\mathbf{x})} q_{N,L}(\mathbf{y})^2 \geq L^{\lambda_2 \log L}$.

Then, there exists $\lambda > 0$ such that $\mathbb{P}\left(\|U_{N,L} - \mathbf{x}\| \geq \frac{\log L}{\sqrt{L}}\right) \leq L^{-\lambda \log L}$ holds for any (L, N) satisfying the same conditions.

Lemma 3 will be used to prove that Condition (1) of Proposition 1 holds in certain cases. Its proof heavily relies on concavity in order to extend a local bound globally.

Lemma 3. *Let f be a concave continuous function on a convex domain $\mathcal{C} \subseteq \mathbb{R}^k$ that has a maximum on $\mathcal{C}$ at $\mathbf{y}$. Assume furthermore that there exist two positive real constants ρ and η such that for every $\mathbf{x} \in \overline{B}(\mathbf{y}, \rho) \cap \mathcal{C}$ the following inequality holds:*

$$f(\mathbf{x}) \leq f(\mathbf{y}) - \eta\|\mathbf{x} - \mathbf{y}\|^2.$$

Then for every positive real $r \leq \rho$, for every $\mathbf{x} \in \mathcal{C}$ such that $\|\mathbf{x} - \mathbf{y}\| > r$ we have $f(\mathbf{x}) \leq f(\mathbf{y}) - \eta\, r^2$.

Finally, we will use the following bounds for $p_L(\mathbf{y})$ and $q_{N,L}(\mathbf{y})$ in terms of $p_L(\mathbf{x})$, which are obtained using basic computations.

Lemma 4. *Let $\mathbf{x} \in \tilde{\mathcal{P}}$ and let $\mathbf{y} \in \overline{B}(\mathbf{x}, k/L) \cap \mathcal{P}$. There exist two positive constants κ_1 and κ_2 such that $\kappa_1 p_L(\mathbf{x}) \leq p_L(\mathbf{y}) \leq 2\kappa_2 p_L(\mathbf{x})$ and*

$$1 - \exp(-\kappa_1 N\, p_L(\mathbf{x})) \leq q_{N,L}(\mathbf{y}) \leq 1 - \exp(-\kappa_2 N\, p_L(\mathbf{x})).$$

4.2 Proof for Range (a): $\ell \leq \ell_0$

In this section, we are in the case where $\ell \leq \ell_0 = \frac{-k}{\sum_{i\in[k]} \log p_i}$. Our goal is to apply Proposition 1 for $\mathbf{x} = \overline{\mathbf{x}} = (\frac{1}{k}, \ldots, \frac{1}{k})$, with $\ell^- = -\infty$ and $\ell^+ = \ell_0$. Observe that $N^{\ell H(\overline{\mathbf{x}})} = k^L$ and that, by Lemma 4, for any $\mathbf{y} \in \overline{B}(\mathbf{x}, k/L)$ we have

$$q_{N,L}(\mathbf{y}) \geq 1 - \exp(-\kappa_1 N p_L(\mathbf{x})) = 1 - \exp\left(-\kappa_1 N^{1+\frac{\ell}{k}\sum_{i\in[k]}\log p_i}\right).$$

As $\ell \leq \ell_0$, we have $1 + \frac{\ell}{k}\sum_{i\in[k]} \log p_i \geq 0$ and thus $q_{N,L}(\mathbf{y}) \geq 1 - e^{-\kappa_1}$.

To verify Condition (1) of Proposition 1, we rely on the following result on the entropy function H, which can be obtained by standard techniques of analysis in several variables:

Lemma 5. *The exists a neighborhood $\mathcal{V}_{\overline{\mathbf{x}}}$ of $\overline{\mathbf{x}} = (\frac{1}{k}, \ldots, \frac{1}{k})$ such that, for every $\mathbf{x} \in \mathcal{P} \cap \mathcal{V}_{\overline{\mathbf{x}}}$ the following inequalities hold:*

$$\log k - k\|\mathbf{x} - \overline{\mathbf{x}}\|^2 \leq H(\mathbf{x}) \leq \log k - \frac{k\|\mathbf{x} - \overline{\mathbf{x}}\|^2}{3}. \tag{3}$$

When L is sufficiently large, $\overline{B}(\overline{\mathbf{x}}, \log L/\sqrt{L}) \subseteq \mathcal{V}_{\overline{\mathbf{x}}}$ of Lemma 5. So we can apply Lemma 3 with $\mathbf{y} = \overline{\mathbf{x}}$ and $r = \log L/\sqrt{L}$, to obtain that for every $\mathbf{x} \in \mathcal{B}_L(\overline{\mathbf{x}})$ we have

$$H(\mathbf{x}) \leq H(\overline{\mathbf{x}}) - \frac{\eta \log^2 L}{L} = \log k - \frac{k \log^2 L}{3L}.$$

By Lemma 1, we can bound the number of words in $\mathcal{W}_L(\mathbf{x})$ for $\mathbf{x} \in \mathcal{B}_L(\overline{\mathbf{x}})$:

$$|\mathcal{W}_L(\mathbf{x})| \leq \alpha N^{\ell\, H(\mathbf{x})} \leq \alpha N^{\ell \log k - \ell \frac{k \log^2 L}{3L}} = \alpha k^L\, L^{-k \log L/3}.$$

Observe that $|\mathcal{B}_L \cap \mathcal{F}_L| \leq |\mathcal{F}_L| \leq L^k$, since there are at most L^k compositions of letters for words of A^L. Therefore,

$$\mathbb{E}|\mathcal{W}_L(\mathcal{B}_L(\overline{\mathbf{x}})) \cap \mathcal{S}_{N,L}| \leq |\mathcal{W}_L(\mathcal{B}_L(\overline{\mathbf{x}})) \cap \mathcal{F}_L| \leq \alpha L^k\, k^L\, L^{-k \log L/3}.$$

This proves that Condition (1) holds in our case, since for any $\lambda_1 < \frac{k}{3}$, the right term is at most $q_{N,L}(\mathbf{y}) k^L L^{-\lambda_1 \log L}$, as we saw that $q_{N,L}(\mathbf{y}) \geq 1 - e^{-\kappa_1}$ at the beginning of the section.

Condition (2) trivially holds as we have, for any $\lambda_2 > 0$:

$$N^{\ell H(\mathbf{x})} q_{N,L}(\mathbf{y})^2 \geq (1 - e^{-\kappa_1})^2 k^L \geq L^{\lambda_2 \log L}.$$

We can therefore apply Proposition 1, concluding the proof for range (a).

4.3 Proof for Range (b): $\ell_0 \leq \ell \leq \ell_1$

In this section, we are in the case where $\frac{-k}{\sum_{i\in[k]} \log p_i} \leq \ell \leq \frac{1}{H(\mathbf{p})}$. Our goal is still to apply Proposition 1, but we first need to find the vector $\mathbf{x}$ that concentrates the frequency vectors of the output of our process.

Recall that $K_\ell(\mathbf{x}) = H(\mathbf{x}) + \frac{1}{\ell} + \sum_{i\in[k]} x_i \log p_i$. We have the following bound on the expected number of words of given frequency vectors that appear in $\mathcal{S}_{N,L}$. It is obtained by linearity of the expectation, and by obtaining bounds on $q_{N,L}(\mathbf{x})$ as in Lemma 4.

Lemma 6. *Let $\mathbf{x} \in \mathcal{P}$. The expected cardinality of $\mathcal{S}_{N,L} \cap \mathcal{W}_L(\mathbf{x})$ satisfies*

$$\mathbb{E}\,|\mathcal{S}_{N,L} \cap \mathcal{W}_L(\mathbf{x})| \le 2\alpha N^{\ell \min(H(\mathbf{x}),K_\ell(\mathbf{x}))},$$

with the same α as in Lemma 1.

Recall that $G_\ell(\mathbf{x}) = \min(H(\mathbf{x}), K_\ell(\mathbf{x}))$, so we can rewrite the bound in Lemma 6 into $2\alpha N^{G_\ell(\mathbf{x})}$. We now study $G_\ell(\mathbf{x})$ for the range corresponding to case (b), *i.e.* $\ell_0 \le \ell \le \ell_1$. Recall also that Φ is the mapping defined by $\Phi(t) = \sum_{i\in[k]} p_i^t$.

Lemma 7. *For any ℓ such that $\ell_0 \le \ell \le \ell_1$, as a function on $\mathcal{P}$, the function $G_\ell(\mathbf{x})$ admits a unique maximum at $\mathbf{x}_c = \frac{1}{\Phi(c)}(p_1^c, \dots, p_k^c)$, where $c \in [0,1]$ is the unique solution of $\ell\Phi'(c) + \Phi(c) = 0$. Moreover, $\mathbf{x}_c$ is in the hyperplane defined by the equation $H(\mathbf{x}) = K_\ell(\mathbf{x})$.*

Now that the maximum $\mathbf{x}_c$ is located, we want to provide an upper bound the expect cardinality of $\mathcal{B}_L(\mathbf{x}_c) \cap \mathcal{S}_{N,L}$, to fulfill Condition (1) of Proposition 1. For this, we want to use Lemma 3, and therefore need an upper bound of $G_\ell(\mathbf{x})$ around its maximum $G_\ell(\mathbf{x}_c)$. This is the purpose of Lemma 8 below, whose proof relies on classical analysis of functions of several variables.

Lemma 8. *Let c be the unique solution of $\ell\Phi'(c) + \Phi(c) = 0$. There exists a real constant $\rho > 0$ such that for every $\mathbf{x} \in \overline{B}(\mathbf{x}_c, \rho) \cap \mathcal{P}$, $G_\ell(\mathbf{x}) \le G_\ell(\mathbf{x}_c) - \|\mathbf{x} - \mathbf{x}_c\|^2$.*

We will also need the following technical lemma, to prove that the cardinality of $\mathcal{W}_L(\mathbf{x}_c)$, which is roughly $N^{\ell H(\mathbf{x}_c)}$, is at least some power of N. Its proof consists in studying $\ell \mapsto \ell H(\mathbf{x}_c)$, where c is viewed as a function of ℓ given by the implicit solution of $\ell\Phi'(c) + \Phi(c) = 0$.

Lemma 9. *For ℓ within range (b), let c be the solution of $\ell\Phi'(c) + \Phi(c) = 0$, the quantity $\ell H(\mathbf{x}_c)$ satisfies the following inequalities:*

$$0 < d \le \ell H(\mathbf{x}_c) \le 1, \ \textit{with } d = \frac{k \log k}{-\sum_{i\in[k]} \log p_i}.$$

Since $H(\mathbf{x}_c) = K_\ell(\mathbf{x}_c)$, we have $N^{\ell H(\mathbf{x}_c)} = N^{\ell G_\ell(\mathbf{x}_c)}$. Moreover, by Lemma 4, for any $\mathbf{y} \in \overline{B}(\mathbf{x}_c, k/L)$ we have

$$q_{N,L}(\mathbf{y}) \ge 1 - \exp(-\kappa_1 N p_L(\mathbf{x}_c)) = 1 - \exp\left(-\kappa_1 N^{1+\ell\sum_{i\in[k]} \frac{p_i^c}{\Phi(c)} \log p_i}\right).$$

But $\ell \sum_{i\in[k]} \frac{p_i^c}{\Phi(c)} \log p_i = \ell\frac{\Phi'(c)}{\Phi(c)} = -1$. Therefore, $q_{N,L}(\mathbf{y}) \ge 1 - e^{-\kappa_1}$.

We can now prove that Condition (1) holds. By Lemmas 8 and 3, when L is sufficiently large, for all $\mathbf{x} \in \mathcal{B}_L(\mathbf{x}_c)$ we have $G_\ell(\mathbf{x}) \leq G_\ell(\mathbf{x}_c) - \log^2 L/L$. Thus, by Lemma 6 we have

$$\mathbb{E}|\mathcal{W}_L(\mathbf{x}) \cap \mathcal{S}_{N,L}| \leq 2\alpha N^{\ell G_\ell(\mathbf{x})} \leq 2\alpha N^{\ell G_\ell(\mathbf{x}_c) - \ell \log^2 L/L} = 2\alpha N^{\ell H(\mathbf{x}_c)} L^{-\log L}.$$

Since $|\mathcal{B}_L(\mathbf{x}_c) \cap \mathcal{S}_{N,L}| \leq |\mathcal{F}_L| \leq L^k$, by linearity of the expectation we have

$$\mathbb{E}|\mathcal{W}_L(\mathcal{B}_L(\mathbf{x}_c)) \cap \mathcal{S}_{N,L}| \leq 2\alpha L^k N^{\ell H(\mathbf{x}_c)} L^{-\log L}.$$

As $q_{N,L}(\mathbf{y}) \geq 1 - e^{-\kappa_1}$, Condition (1) holds since

$$\mathbb{E}|\mathcal{W}_L(\mathcal{B}_L(\mathbf{x}_c)) \cap \mathcal{S}_{N,L}| \leq N^{\ell H(\mathbf{x}_c)} q_{N,L}(\mathbf{y}) L^{-\frac{1}{2}\log L}.$$

Condition (2) also holds: by Lemma 9, we have

$$N^{\ell H(\mathbf{x}_c)} q_{N,L}(\mathbf{y})^2 \geq N^d (1 - e^{-\kappa_1})^2 \leq L^{-\log L},$$

since the condition $\ell \leq \ell_1$ implies that N grows exponentially in L. Therefore, we can apply Proposition 1, concluding the proof for range (b).

4.4 Proof for Range (c): $\ell \geq \ell_1$

For this range we cannot only rely on Proposition 1, as when ℓ is very large, N is small towards $\log L$, or even towards L, and the conditions do not hold anymore. We therefore split the proof into several subranges for ℓ.

To provide an upper bound for the expected cardinality of $|\mathcal{B}_L(\mathbf{p}) \cap \mathcal{S}_{N,L}|$, we use the classical large deviation results for the multinomial distribution [7, p. 462] to obtain the following lemma:

Lemma 10. *The following inequality holds:* $\mathbb{P}_L\left(\mathcal{B}_L(\mathbf{p})\right) \leq 2^k\, L^{-\frac{1}{2}\log L}$.

As a consequence, since we generate N random words with the source,

$$\mathbb{E}|\mathcal{B}_L(\mathbf{p}) \cap \mathcal{S}_{N,L}| \leq 2^k N\, L^{-\frac{1}{2}\log L}.$$

Observe also that $p_L(\mathbf{p}) = N^{-\ell H(\mathbf{p})}$, thus by Lemma 4, for $\mathbf{y} \in \mathcal{B}_L(\mathbf{p}, k/L)$,

$$q_{N,L}(\mathbf{y}) \geq 1 - \exp(-\kappa_1 N p_L(\mathbf{p})) = 1 - \exp\left(-\kappa_1 N^{1-\ell H(\mathbf{p})}\right).$$

As $e^{-t} \leq 1 - \frac{t}{2}$ for $t \in [0,1]$, we have $q_{N,L}(\mathbf{y}) \geq \frac{\kappa_1}{2} N^{1-\ell H(\mathbf{p})}$. Thus we have $N^{\ell H(\mathbf{p})} q_{N,L}(\mathbf{y}) \geq \frac{\kappa_1}{2} N$ and Condition (1) of Proposition 1 holds for any $\lambda_1 \in (0, \frac{1}{2})$. Unfortunately, Condition (2) does not always hold, and we have to change the proof when ℓ is too large.

▶ **case** $\boldsymbol{\frac{1}{H(\mathbf{p})} \leq \ell \leq \frac{3}{2H(\mathbf{p})}}$**:** in this case, Condition (2) holds. Indeed,

$$N^{\ell H(\mathbf{p})} q_{N,L}(\mathbf{y})^2 \geq \frac{\kappa_1}{2} N^{2-\ell H(\mathbf{p})} \geq \frac{\kappa_1}{2}\sqrt{N}.$$

But the condition $\ell \leq \frac{3}{2H(\mathbf{p})}$ implies that N grows exponentially in N. Hence, Proposition 1 applies and the result holds for this subrange.

▶ **case $\frac{3}{2H(\mathbf{p})} \leq \ell \leq \ell_2$:** where $\ell_2 = \frac{2}{-\log \lambda}$, for some $\lambda \in (p_{\max}, 1)$. To complete the proof, we have to establish that $|\mathcal{S}_{N,L}|$ is large with very high probability. By Lemma 1, there exists $\mathbf{y} \in \overline{B}(\mathbf{p}, \frac{1}{L})$ such that $|\mathcal{W}_L(\mathbf{y})| \geq \beta L^{(1-k)/2} N^{\ell H(\mathbf{p})}$. By Lemma 4, $p_L(\mathbf{y}) \geq \kappa_1 N^{\ell H(\mathbf{p})}$ and therefore $\mathbb{P}_L(\mathcal{W}_L(\mathbf{y})) \geq \kappa L^{(1-k)/2}$, with $\kappa = \kappa_1 \beta$.

We now consider the process from the start, when the N words of length L are repeatedly generated by the source. Let $Y_{N,L}$ be the random variable that counts the number of words of $\mathcal{W}_L(\mathbf{y})$ generated during this process. The random variable $Y_{N,L}$ is distributed as a binomial distribution of coefficients N and $\mathbb{P}_L(\mathcal{W}_L(\mathbf{y}))$: $\mathbb{E}[Y_{N,L}] = N\mathbb{P}_L(\mathcal{W}_L(\mathbf{y}))$ and there is concentration around the mean, by Chernoff-Hoeffding inequality:

$$\mathbb{P}\left(Y_{N,L} \leq \frac{\kappa}{2} L^{(1-k)/2} N\right) \leq \exp\left(\frac{-\kappa^2}{2} L^{1-k} N\right) \leq L^{-\log L}, \qquad (4)$$

for L sufficiently large, since N is exponential in L when $\ell \leq \ell_2$. In the process of repeatedly generating N words, a word u is called a $\mathbf{y}$-*duplicate* if $\mathbf{freq}(u) = \mathbf{y}$ and u has already been generated. Let $D_{N,L}$ be the random variable that counts the number of $\mathbf{y}$-duplicates. We have $|\mathcal{S}_{N,L} \cap \mathcal{W}_L(\mathbf{y})| = Y_{N,L} - D_{N,L}$. If we only look at what happens inside $\mathcal{W}_L(\mathbf{y})$, we are considering the process of choosing $Y_{N,L}$ times an element of $\mathcal{W}_L(\mathbf{y})$ uniformly at random, as they all have the same probability of being generated. We will use the following classical lemma.

Lemma 11. *Let E be a set of cardinality $n \geq 1$. Let $D_E(n)$ denote the number of duplicates obtained when generating m times an element of E uniformly at random. Then*

$$\mathbb{E}[D_E(n)] \leq \frac{m^2}{2n}.$$

Since $Y_{N,L} \leq N$, the expected number of duplicates in $\mathcal{S}_{N,L}$ satisfies

$$\mathbb{E}[D_{N,L}] \leq \frac{N^2}{2|\mathcal{W}_L(\mathbf{y})|} \leq \frac{L^{(k-1)/2}}{2\beta} N^{2-\ell H(\mathbf{p})} \leq \frac{L^{(k-1)/2}}{2\beta} \sqrt{N}.$$

By Markov inequality, as N is exponential in L, we get that for any positive ν

$$\mathbb{P}\left(D_{N,L} \geq L^{2\nu \log L} \sqrt{N}\right) \leq \mathbb{P}\left(D_{N,L} \geq \frac{L^{(k-1)/2}}{2\beta} L^{\nu \log L} \sqrt{N}\right) \leq L^{-\nu \log L}. \quad (5)$$

Equations 4 and 5 ensure by the union bound that

$$\mathbb{P}\left(Y_{N,L} - D_{N,L} \leq \frac{\kappa}{2} L^{(1-k)/2} N - L^{2\nu \log L} \sqrt{N}\right) \leq L^{-\log L} + L^{-\nu \log L}.$$

Recall that $Y_{N,L} - D_{N,L} = |\mathcal{W}(\mathbf{y}) \cap \mathcal{S}_{N,L}|$. As N is exponential in L, for any positive constant μ we have

$$\mathbb{P}\left(|\mathcal{W}(\mathbf{y}) \cap \mathcal{S}_{N,L}| \leq L^{-\mu \log L} N\right) \leq 2L^{-\nu \log L}.$$

Such a precise estimation of the cardinality of $\mathcal{W}(\mathbf{y}) \cap \mathcal{S}_{N,L}$ can be used as a substitute for Condition (2) of Proposition 1, concluding the proof for this subrange.

▶ **case $\boldsymbol{\ell \geq \ell_2}$:** In this case, we simply rely on this classical result deduced from the Birthday Paradox problem, to establish that with very high probability, there are no duplicates in $\mathcal{S}_{N,L}$.

Lemma 12. *If $\ell \geq \ell_2$, then the probability that $|\mathcal{S}_{N,L}| < N$ is exponentially small in L.*

As a consequence, this proves our result for $\ell \geq \ell_2$: if the N elements generated by the source are pairwise distinct, then $U_{N,L}$ is distributed as a random element of the source. By Lemma 12, this happen with probability at least $1 - L^{-\log L}$.

5 Conclusions

In this article, we exhibited two thresholds for the typical frequency vector of a word in the partial word collector problem, for a memoryless source of parameter $\mathbf{p}$. We were able to establish that with high probability, it resembles either the uniform distribution, or a word generated by the source, or some vector in between, which we fully characterize.

We want to make the observation that, though two words with the same frequency vector have the same probability to be the result $U_{N,L}$ of our process, $U_{N,L}$ is not distributed as the output of a memoryless source: the computations for 3 words of length 2 on $\{a, b\}$ show that the probability that $U_{N,L}$ ends by a is not the same if we condition by starting by a or by b.

A natural continuation of this work, is to study the case where the N words are not independent anymore, but are the factors of length L of a random word of length $N + L - 1$. As shown for instance in [4], the correlation between the factors is small, so we expect a similar result, even if it is still ongoing work.

Another possible extension would be to study a similar word collection process when words are not produced by a memoryless source; say, using a Markov source instead. One would expect at least equivalents of regimes (a) and (c) in our theorem, though regime (c) could have asymptotic frequencies that are not uniform (as the support of the distribution may not be A^L); and any intermediate regime(s) could be much more difficult to determine.

Acknowledgments. The authors are grateful to Arnaud Carayol for his precious help when preparing this article, and an anonymous reviewer for suggesting the promising alternative α-parametrization of the problem.

References

1. Du Boisberranger, J., Gardy, D., Ponty, Y.: The weighted words collector. In: AOFA - 23rd International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms - 2012, pp. 243–264. DMTCS (2012)
2. Dubhashi, D., Ranjan, D.: Balls and bins: a study in negative dependence. Random Struct. Algorithms **13**(2), 99–124 (1998)
3. Duchon, P., Nicaud, C., Pivoteau, C.: Gapped pattern statistics. In: Kärkkäinen, J., Radoszewski, J., Rytter, W. (eds.) 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, 4–6 July 2017, Warsaw, Poland. LIPIcs, vol. 78, pp. 21:1–21:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
4. Gheorghiciuc, I., Ward, M.D.: On correlation polynomials and subword complexity. In: Discrete Mathematics and Theoretical Computer Science, DMTCS Proceedings, vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07), January 2007
5. MacKay, D.J.: Information Theory, Inference and Learning Algorithms. Cambridge University Press, Cambridge (2003)
6. Rubinchik, M., Shur, A.M.: The number of distinct subpalindromes in random words. Fundam. Inform. **145**(3), 371–384 (2016)
7. Van Der Vaart, A.W., Wellner, J.A.: Weak convergence. In: Van Der Vaart, A.W., Wellner, J.A. (eds.) Weak Convergence and Empirical Processes, pp. 16–28. Springer, New York (1996). https://doi.org/10.1007/978-1-4757-2545-2_3

Constructive Ramsey Numbers for Loose Hyperpaths

Andrzej Dudek[1(✉)] and Andrzej Ruciński[2]

[1] Department of Mathematics, Western Michigan University, Kalamazoo, MI, USA
andrzej.dudek@wmich.edu

[2] Department of Discrete Mathematics, Adam Mickiewicz University, Poznań, Poland
rucinski@amu.edu.pl

Abstract. For positive integers k and ℓ, a k-uniform hypergraph is called a *loose path of length* ℓ, and denoted by $P_\ell^{(k)}$, if its vertex set is $\{v_1, v_2, \ldots, v_{(k-1)\ell+1}\}$ and the edge set is $\{e_i = \{v_{(i-1)(k-1)+q} : 1 \leq q \leq k\},\ i = 1, \ldots, \ell\}$, that is, each pair of consecutive edges intersects on a single vertex. Let $R(P_\ell^{(k)}; r)$ be the *multicolor Ramsey number of a loose path* that is the minimum n such that every r-edge-coloring of the complete k-uniform hypergraph $K_n^{(k)}$ yields a monochromatic copy of $P_\ell^{(k)}$. In this note we are interested in *constructive* upper bounds on $R(P_\ell^{(k)}; r)$ which means that on the cost of possibly enlarging the order of the complete hypergraph, we would like to efficiently find a monochromatic copy of $P_\ell^{(k)}$ in every coloring. In particular, we show that there is a constant $c > 0$ such that for all $k \geq 2$, $\ell \geq 3$, $2 \leq r \leq k-1$, and $n \geq k(\ell+1)r(1+\ln(r))$, there is an algorithm such that for every r-edge-coloring of the edges of $K_n^{(k)}$, it finds a monochromatic copy of $P_\ell^{(k)}$ in time at most cn^k.

1 Introduction

For positive integers $k \geq 2$ and $\ell \geq 0$, a k-uniform hypergraph is called a *loose path of length* ℓ, and denoted by $P_\ell^{(k)}$, if its vertex set is $\{v_1, v_2, \ldots, v_{(k-1)\ell+1}\}$ and the edge set is $\{e_i = \{v_{(i-1)(k-1)+q} : 1 \leq q \leq k\},\ i = 1, \ldots, \ell\}$, that is, for $\ell \geq 2$, each pair of consecutive edges intersects on a single vertex (see Fig. 1), while for $\ell = 0$ and $\ell = 1$ it is, respectively, a single vertex and an edge. For $k = 2$ the loose path $P_\ell^{(2)}$ is just a (graph) path on $\ell + 1$ vertices.

Let H be a k-uniform hypergraph and $r \geq 2$ be an integer. The *multicolor Ramsey number* $R(H; r)$ is the minimum n such that every r-edge-coloring of the complete k-uniform hypergraph $K_n^{(k)}$ yields a monochromatic copy of H.

A. Dudek—supported in part by Simons Foundation Grant #522400.
A. Ruciński—supported in part by the Polish NSC grant 2014/15/B/ST1/01688.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 427–436, 2018.
https://doi.org/10.1007/978-3-319-77404-6_31

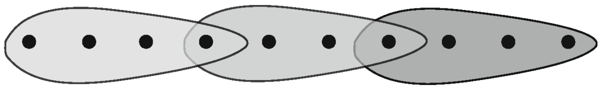


Fig. 1. A 4-uniform loose path $P_3^{(4)}$.

For graphs, determining the Ramsey number $R(P_\ell^{(2)}, r)$ is a well-known problem that attracted a lot of attention. It was shown by Gerencsér and Gyárfás [6] that

$$R(P_\ell^{(2)}, 2) = \left\lfloor \frac{3\ell+1}{2} \right\rfloor. \tag{1}$$

For three colors Figaj and Łuczak [5] proved that $R(P_\ell^{(2)}, 3) \approx 2\ell$. Soon after, Gyárfás et al. [7,8] determined this number exactly, showing that for all sufficiently large ℓ

$$R(P_\ell^{(2)}, 3) = \begin{cases} 2\ell+1 & \text{for even } \ell, \\ 2\ell & \text{for odd } \ell, \end{cases} \tag{2}$$

as conjectured earlier by Faudree and Schelp [4]. For $r \geq 4$ much less is known. A celebrated Turán-type result of Erdős and Gallai [3] implies that

$$R(P_\ell^{(2)}, r) \leq r\ell. \tag{3}$$

Recently, this was slightly improved by Sárközy [9] and, subsequently, by Davies et al. [1] who showed that for all sufficiently large ℓ,

$$R(P_\ell^{(2)}; r) \leq (r - 1/4)(\ell + 1). \tag{4}$$

In this note we are mostly interested in *constructive* bounds which means that on the cost of possibly enlarging the order of the complete hypergraph, we would like to efficiently find a monochromatic copy of a target hypergraph F in every coloring. Clearly, by examining all copies of F in $K_n^{(k)}$ for $n \geq R(F; r)$, we can always find a monochromatic one in time $O(n^{|V(F)|})$. Hence, we are interested in complexity not depending on F, preferably $O(n^k)$. Given a k-graph F, a constant $c > 0$ and integers r and n, we say that a property $\mathcal{R}(F, r, c, n)$ holds if there is an algorithm such that for every r-edge-coloring of the edges of $K_n^{(k)}$, it finds a monochromatic copy of F in time at most cn^k. For graphs, a constructive result of this type can be deduced from the proof of Lemma 3.5 in Dudek and Prałat [2].

Theorem 1 ([2]). *There is a constant $c > 0$ such that for all $\ell \geq 3$, $r \geq 2$, and $n \geq 2^{r+1}\ell$, property $\mathcal{R}(P_\ell^{(2)}, r, c, n)$ holds.*

Our goal is to obtain similar constructive results for loose hyperpaths. However, to have a reference point we first state, without proof, a general (non-constructive) upper bound, obtained iteratively for all $k \geq 2$, starting from the Erdős-Gallai bound (3).

Theorem 2. *For all $k \geq 2$, $\ell \geq 3$, and $r \geq 2$ we have $R(P_\ell^{(k)}; r) \leq (k-1)\ell r$.*

One can show that Theorem 2 can be improved for $r = 2$ or for large ℓ. For $r = 2$, using (1) instead of (3) at the base step, one gets, for $k \geq 3$,

$$R(P_\ell^{(k)}; 2) \leq (2k - 5/2)\ell. \tag{5}$$

For large ℓ, using (2) instead of (3), we obtain for $r = 3$ that

$$R(P_\ell^{(k)}; 3) \leq (3k - 4)\ell,$$

and for $r \geq 4$, by (4),

$$R(P_\ell^{(k)}; r) \leq (k-1)\ell r - \ell/4.$$

By replacing the Erdős-Gallai bound (3) with the assumption on n given in Theorem 1, the proof of Theorem 2 can be adapted to yield a constructive result.

Theorem 3. *There is a constant $c > 0$ such that for all $k \geq 2$, $\ell \geq 3$, $r \geq 2$, and $n \geq 2^{r+1}\ell + (k-2)\ell r$, property $\mathcal{R}(P_\ell^{(k)}, r, c, n)$ holds.*

Our main constructive bound (valid only for $r \leq k$) utilizes a more sophisticated algorithm.

Theorem 4. *There is a constant $c > 0$ such that for all $k \geq 2$, $\ell \geq 3$, $2 \leq r \leq k$, and $n \geq k(\ell+1)r\left(1 + \frac{1}{k-r+1} + \ln\left(1 + \frac{r-2}{k-r+1}\right)\right)$, property $\mathcal{R}(P_\ell^{(k)}, r, c, n)$ holds. For $r = 2$, the bound on n can be improved to $n \geq (2k-2)\ell + k$.*

Note that for $r = 2$ the lower bound on n in Theorem 4 is very close to that in (5). For $r = k \geq 3$ the bound assumes a simple form

$$n \geq k^2(\ell + 1)(2 + \ln(k-1)).$$

Furthermore, for $r \leq k-1$, one can show that

$$\frac{1}{k-r+1} + \ln\left(1 + \frac{r-2}{k-r+1}\right) \leq \ln\left(1 + \frac{r-1}{k-r}\right)$$

which yields the following corollary.

Corollary 1. *There is a constant $c > 0$ such that for all $k \geq 3$, $\ell \geq 3$, $3 \leq r \leq k-1$, and $n \geq k(\ell+1)r\left(1 + \ln\left(1 + \frac{r-1}{k-r}\right)\right)$, property $\mathcal{R}(P_\ell^{(k)}, r, c, n)$ holds.*

We can further replace the lower bound on n by (slightly weaker but simpler) $n \geq k(\ell+1)r(1 + \ln r)$.

Observe that in several instances the lower bound on n in Theorem 4 (and also in Corollary 1) is significantly better (that means smaller) than the one in Theorem 3 (for example for large k and $k/2 \leq r \leq k$). On the other hand, for some instances the bounds in Theorems 3 and 4 are basically the same. For example, for fixed r, large k and $\ell \geq k$ the lower bound is $k\ell r + o(k\ell)$. This also matches asymptotically the bound in Theorem 2.

In this note we only present the proof of Theorem 4.

2 Proof of Theorem 4

Given integers k and $2 \leq m \leq k$, and disjoint sets of vertices $W_1, \ldots, W_{m-1}, V_m$, *an m-partite complete k-graph* $K^{(k)}(W_1, \ldots, W_{m-1}, V_m)$ consists of all k-tuples of vertices with exactly one element in each W_i, $i = 1, \ldots, m-1$, and $k-m+1$ elements in V_m. Note that if $|W_i| \geq \ell$, $i = 1, \ldots, m-1$, and $|V_m| \geq \ell(k-m)+1$ for $m \leq k-1$ (or $|V_m| \geq \ell$ for $m = k$), then $K^{(k)}(W_1, \ldots, W_{m-1}, V_m)$ contains $P_\ell^{(k)}$.

We now give a description of the algorithm. As an input there is an r-coloring of the edges of the complete k-graph $K_n^{(k)}$. The algorithm consists of $r-1$ implementations of the depth first search (DFS) subroutine, each round exploring the edges of one color only and either finding a monochromatic copy of $P_\ell^{(k)}$ or decreasing the number of colors present on a large subset of vertices, until after the $(r-1)$st round we end up with a monochromatic complete r-partite subgraph, large enough to contain a copy of $P_\ell^{(k)}$.

During the ith round, while trying to build a copy of the path $P_\ell^{(k)}$ in the ith color, the algorithm selects a subset $W_{i,i}$ from a set of still available vertices $V_i \subseteq V$ and, by the end of the round, creates trash bins S_i and T_i. The search for $P_\ell^{(k)}$ is realized by a DFS process which maintains a working path P (in the form of a sequence of vertices) whose endpoints are either extended to a longer path or otherwise put into $W_{i,i}$. The round is terminated whenever P becomes a copy of $P_\ell^{(k)}$ or the size of $W_{i,i}$ reaches certain threshold, whatever comes first. In the latter case we set $S_i = V(P)$.

To better depict the extension process, we introduce the following terminology. An edge of $P_\ell^{(k)}$ is called *pendant* if it contains at most one vertex of degree two. The vertices of degree one, belonging to the pendant edges of $P_\ell^{(k)}$ are called *pendant.* In particular, in $P_1^{(k)}$ all its k vertices are pendant. For convenience, the unique vertex of the path $P_0^{(k)}$ is also considered to be pendant. Observe that for $t \geq 0$, to extend a copy P of $P_t^{(k)}$ to a copy of $P_{t+1}^{(k)}$ one needs to add a new edge which shares exactly one vertex with P and that vertex has to be pendant in P. Our algorithm may also come across a situation when $P = \emptyset$, that is, P has no vertices at all. Then by an extension of P we mean any edge whatsoever.

The sets $W_{i,i}$ have a double subscript, because they are updated in the later rounds to $W_{i,i+1}$, $W_{i,i+2}$, and so on, until at the end of the $(r-1)$st round (unless a monochromatic $P_\ell^{(k)}$ has been found) one obtains sets $W_i := W_{i,r-1}$, $i = 1, \ldots, r-1$, a final trash set $T = \bigcup_{i=1}^{r-1} T_i \cup \bigcup_{i=1}^{r-1} S_i$ and the remainder set $V_r = V \setminus (\bigcup_{i=1}^{r-1} W_i \cup T)$ such that all k-tuples of vertices in $K^{(k)}(W_1, \ldots, W_{r-1}, V_r)$ are of color r. As an input of the ith round we take sets $W_{j,i-1}$, $j = 1, \ldots, i-1$, and V_{i-1}, inherited from the previous round, and rename them to $W_{j,i}$, $j = 1, \ldots, i-1$, and V_i. We also set $T_i = \emptyset$ and $P = \emptyset$, and update all these sets dynamically until the round ends.

Now come the details. For $1 \leq i \leq r-1$, let

$$\tau_i = \begin{cases} (i-1)\left(\frac{\ell}{k-r+1} + \frac{\ell+1}{k-r+2} + \cdots + \frac{\ell+1}{k-i}\right) & \text{if } 1 \leq i \leq r-2, \\ (r-2)\frac{\ell}{k-r+1} & \text{if } i = r-1, \end{cases} \tag{6}$$

and

$$t_i = \tau_i + 2(i-1).$$

Note that τ_i is generally not an integer. It can be easily shown that for all $2 \leq r \leq k$ and $1 \leq i \leq r-1$

$$\tau_i \leq (i-1)(\ell+1)\left(\frac{1}{k-r+1} + \ln\left(1 + \frac{r-2}{k-r+1}\right)\right). \tag{7}$$

Before giving a general description of the ith round, we deal separately with the 1st and 2nd round.

Round 1. Set $V_1 = V$, $W_{1,1} = \emptyset$, and $P = \emptyset$. Select an arbitrary edge e of color one (say, *red*), add its vertices to P (in any order), reset $V_1 := V_1 \backslash e$, and try to extend P to a red copy of $P_2^{(k)}$. If successful, we appropriately enlarge P, diminish V_1, and try to further extend P to a red copy of $P_3^{(k)}$. This procedure is repeated until finally we either find a red copy of $P_\ell^{(k)}$ or, otherwise, end up with a red copy P of $P_t^{(k)}$, for some $1 \leq t \leq \ell - 1$, which cannot be extended any more. In the latter case we shorten P by moving all its pendant vertices to $W_{1,1}$ and try to extend the remaining red path again. When $t \geq 2$, the new path has $t-2$ edges. If $t = 2$, P becomes a single vertex path $P_0^{(k)}$, while if $t = 1$, it becomes empty.

Let us first consider the simplest but instructive case $r = 2$ in which only one round is performed. If at some point $P = \emptyset$ and cannot be extended (which means there are no red edges within V_1), then we move $\ell - |W_{1,1}|$ arbitrary vertices from $V_1 = V \backslash W_{1,1}$ to $W_{1,1}$ and stop. Otherwise, we terminate Round 1 as soon as

$$|W_{1,1}| \geq \ell.$$

At that moment, no edge of $K^{(k)}(W_{1,1}, V_1)$ is red (so, all of them must be, say, blue). Moreover, since the size of $W_{1,1}$ increases by increments of at most $2(k-1)$, we have

$$\ell \leq |W_{1,1}| \leq \ell + 2(k-1) - 1,$$

and, consequently,

$$|V_1| = n - |W_{1,1}| - |V(P)| \geq n - \ell - 2(k-1) + 1 - |V(P_{\ell-1}^{(k)})| \geq \ell(k-2) + 1$$

by our bound on n. This means that the completely blue copy of $K^{(k)}(W_{1,1}, V_1)$ is large enough to contain a copy of $P_\ell^{(k)}$.

When $r \geq 3$, there are still more rounds ahead during which the set $W_{1,1}$ will be cut down, so we need to ensure it is large enough to survive the entire process.

To this end we alter the stopping rule as follows. If at some point $P = \emptyset$ and cannot be extended, we move $\lceil (k-1)\tau_2 \rceil + \ell + 1 - |W_{1,1}|$ arbitrary vertices from $V_1 = V \backslash W_{1,1}$ to $W_{1,1}$ and stop. Otherwise, we terminate Round 1 as soon as

$$|W_{1,1}| \geq (k-1)\tau_2 + \ell + 1. \tag{8}$$

Since the size of $W_{1,1}$ increases by increments of at most $2(k-1)$ and the R-H-S of (8) is not necessarily integer, we also have

$$|W_{1,1}| \leq (k-1)\tau_2 + \ell + 1 + 2(k-1). \tag{9}$$

Finally, we set $S_1 := P$, $T_1 = \emptyset$ for mere convenience, and $V_1 := V \setminus (W_{1,1} \cup S_1 \cup T_1)$. Note that $|S_1| \leq |V(P_{\ell-1}^{(k)})| = (\ell-1)(k-1)+1$. Also, it is important to realize that no edge of $K^{(k)}(W_{1,1}, V_1)$ is colored red.

Round 2. We begin with resetting $W_{1,2} := W_{1,1}$ and $V_2 := V_1$, and setting $P := \emptyset$, $W_{2,2} = \emptyset$, and $T_2 := \emptyset$. In this round only the edges of color two (say, blue) belonging to $K^{(k)}(W_{1,2}, V_2)$ are considered. Let us denote the set of these edges by E_2. We choose an arbitrary edge $e \in E_2$, set $P = e$, and try to extend P to a copy of $P_2^{(k)}$ in E_2 but only in such a way that the vertex of e belonging to $W_{1,2}$ remains of degree one on the path. Then, we try to extend P to a copy of $P_3^{(k)}$ in E_2, etc., always making sure that the vertices in $W_{1,2}$ are of degree one. Eventually, either we find a blue copy of $P_\ell^{(k)}$ or end up with a blue copy P of $P_t^{(k)}$, for some $1 \leq t \leq \ell - 1$, which cannot be further extended. We move the pendant vertices of P belonging to $W_{1,2}$ to the trash set T_2, while the remaining pendant vertices of P go to $W_{2,2}$. Then we try to extend the shortened path again.

We terminate Round 2 as soon as $P = \emptyset$ cannot be extended or

$$|W_{2,2}| \geq (k-2)\tau_2.$$

In the former case we move $\lceil (k-2)\tau_2 \rceil - |W_{2,2}|$ arbitrary vertices from V_2 to $W_{2,2}$. Note that at the end of this round

$$|W_{2,2}| \leq (k-2)\tau_2 + 2(k-2). \tag{10}$$

We set $S_2 := V(P)$ and $V_2 := V \backslash (W_{1,2} \cup W_{2,2} \cup S_2 \cup T_2)$. Observe that no edge of $K^{(k)}(W_{1,2}, W_{2,2}, V_2)$ is red or blue. We will now show that

$$|T_2| \leq t_2 \qquad \text{and} \qquad |W_{1,2}| \geq (k-2)\tau_2. \tag{11}$$

First observe that

$$|W_{1,1}| \leq |W_{1,2}| + |T_2| + \ell - 1. \tag{12}$$

Indeed, at the end of this round $W_{1,1}$ is the union of $W_{1,2} \cup T_2$ and the vertices in $V(P) \cap W_{1,2}$ that were moved to S_2. Since $|V(P) \cap W_{1,2}| \leq \ell - 1$, (12) holds.

Also note that each vertex in T_2 can be matched with a set of $k-2$ or $k-1$ vertices in $W_{2,2}$, and all these sets are disjoint. Consequently,

$$|W_{2,2}| \geq (k-2)|T_2|. \tag{13}$$

Inequality (13) immediately implies that

$$|T_2| \overset{(13)}{\leq} \frac{1}{k-2}|W_{2,2}| \overset{(10)}{\leq} \tau_2 + 2 = t_2.$$

Furthermore,

$$(k-1)\tau_2 + \ell + 1 \overset{(8)}{\leq} |W_{1,1}| \overset{(12)}{\leq} |W_{1,2}| + |T_2| + \ell - 1 \leq |W_{1,2}| + \tau_2 + \ell + 1,$$

completing the proof of (11).

From now on we proceed inductively. Assume that $i \geq 3$ and we have just finished round $i-1$ constructing so far, for each $1 \leq j \leq i-1$, sets S_j, T_j, and $W_{j,i-1}$, satisfying

$$|W_{j,i-1}| \geq \frac{k-i+1}{i-2}\tau_{i-1}, \tag{14}$$

$|S_{i-1}| \leq |V(P^{(k)}_{\ell-1})|$, and $|T_{i-1}| \leq t_{i-1}$, and the residual set

$$V_{i-1} = V \setminus \bigcup_{j=1}^{i-1}(W_{j,i-1} \cup S_j \cup T_j)$$

such that $K^{(k)}(W_{1,i-1}, \ldots, W_{i-1,i-1}, V_{i-1})$ contains no edge of color $1, 2, \ldots,$ or $i-1$.

Round i, $3 \leq i \leq r-1$. We begin the ith round by resetting $W_{1,i} := W_{1,i-1}, \ldots, W_{i-1,i} := W_{i-1,i-1}$, and $V_i := V_{i-1}$, and setting $P := \emptyset$, $W_{i,i} := \emptyset$, and $T_i := \emptyset$. We consider only edges of color i in $K^{(k)}(W_{1,i}, \ldots, W_{i-1,i}, V_i)$. Let us denote the set of such edges by E_i.

As in the previous steps we are trying to extend the current path P using the edges of E_i, but only in such a way that the vertices of degree two in P belong to V_i. When an extension is no longer possible and $P \neq \emptyset$, we move the pendant vertices of P belonging to $\bigcup_{j=1}^{i-1} W_{j,i}$ to the trash set T_i, while the remaining pendant vertices of P go to $W_{i,i}$ (see Fig. 2). Then we try to extend the shortened path. We terminate the ith round as soon as $P = \emptyset$ cannot be extended or

$$|W_{i,i}| \geq \frac{k-i}{i-1}\tau_i.$$

In the former case we move $\lceil \frac{k-i}{i-1}\tau_i \rceil - |W_{i,i}|$ vertices from V_i to $W_{i,i}$. In the latter case, set $S_i := V(P)$. This yields that

$$|W_{i,i}| \leq \frac{k-i}{i-1}\tau_i + 2(k-i). \tag{15}$$

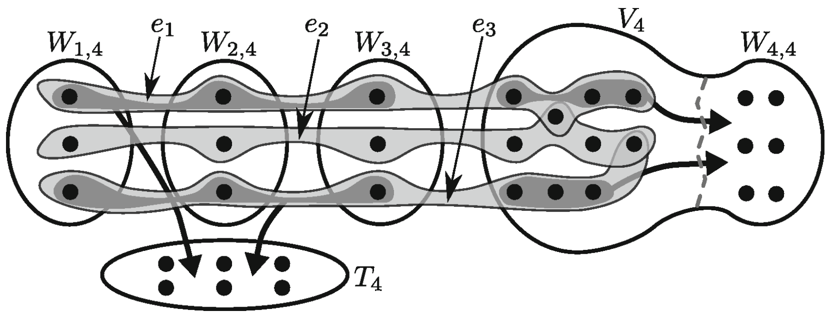


Fig. 2. Applying the algorithm to a 7-uniform hypergraph. Here $i = 4$ and path P, which consists of edges e_1, e_2, and e_3, cannot be extended. Therefore, the vertices in $V(P) \cap (W_{1,4} \cup W_{2,4} \cup W_{3,4})$ are moved to the trash bin T_4 and the pendant vertices in $V_4 \cap (e_1 \cup e_3)$ are moved to $W_{4,4}$.

Similarly as in (12) and (13) notice that for all $1 \le j \le i-1$

$$|W_{j,i-1}| \le |W_{j,i}| + \frac{|T_i|}{i-1} + \ell - 1 \tag{16}$$

and

$$|T_i| \le \frac{i-1}{k-i}|W_{i,i}| \le \tau_i + 2(i-1) = t_i. \tag{17}$$

Thus,

$$\frac{k-i+1}{i-2}\tau_{i-1} \overset{(14)}{\le} |W_{j,i-1}| \overset{(16),(17)}{\le} |W_{j,i}| + \frac{\tau_i}{i-1} + 2 + \ell - 1 = |W_{j,i}| + \frac{\tau_i}{i-1} + \ell + 1$$

and, since also

$$\frac{k-i+1}{i-2}\tau_{i-1} \overset{(6)}{=} \frac{k-i+1}{i-1}\tau_i + \ell + 1,$$

we get

$$|W_{j,i}| \ge \frac{k-i}{i-1}\tau_i. \tag{18}$$

Consequently, when the ith round ends, we have (18) for all $1 \le j \le i$. We also have $|S_i| \le |V(P^{(k)}_{\ell-1})|$, $|T_i| \le t_i$, and $V_i = V \setminus \bigcup_{j=1}^{i}(W_{j,i} \cup S_j \cup T_j)$ such that $K^{(k)}(W_{1,i}, \ldots, W_{i-1,i}, W_{i,i}, V_i)$ has no edges of color $1, 2, \ldots,$ or i.

In particular, when the $(r-1)$st round is finished, we have, for each $1 \le j \le r-1$,

$$|W_{j,r-1}| \ge \frac{k-r+1}{r-2}\tau_{r-1}, \tag{19}$$

$|S_{r-1}| \le |V(P^{(k)}_{\ell-1})|$ and $|T_{r-1}| \le t_{r-1}$. Set $W_j := W_{j,r-1}$, $j = 1, \ldots, r-1$, and $V_r := V \setminus \bigcup_{j=1}^{r-1}(W_j \cup S_j \cup T_j)$ and observe that $K^{(k)}(W_1, \ldots, W_{r-1}, V_r)$ has only edges of color r.

By (19), for each $1 \le j \le r-1$

$$|W_j| \overset{(19)}{\ge} \frac{k-r+1}{r-2}\tau_{r-1} \overset{(6)}{=} \ell.$$

Now we are going to show that $|V_r| \ge \ell(k-r+1)$ which will complete the proof as this bound yields a monochromatic copy of $P_\ell^{(k)}$ inside $K^{(k)}(W_1,\dots,W_{r-1},V_r)$. (Actually for $r \le k-1$ it suffices to show that $|V_r| \ge \ell(k-r)+1$.)

First observe that

$$|W_{1,1}| + \cdots + |W_{r-2,r-2}| \ge |W_1| + \cdots + |W_{r-2}| + |T_1| + \cdots + |T_{r-1}|. \tag{20}$$

This is easy to see, since during the process

$$W_{i,i} \supseteq W_{i,r-1} \cup (W_{i,i} \cap (T_{i+1} \cup \cdots \cup T_{r-1})).$$

Also,

$$\begin{aligned}|W_{1,1}| &\overset{(9)}{\le} (k-1)\tau_2 + 2(k-1) + \ell + 1\\ &\overset{(7)}{\le} (k-1)(\ell+1)\left(\frac{1}{k-r+1} + \ln\left(1+\frac{r-2}{k-r+1}\right)\right) + 2(k-1) + \ell + 1\end{aligned}$$

and, for $2 \le i \le r-1$,

$$\begin{aligned}|W_{i,i}| &\overset{(15)}{\le} \frac{k-i}{i-1}\tau_i + 2(k-i)\\ &\overset{(7)}{\le} (k-i)(\ell+1)\left(\frac{1}{k-r+1} + \ln\left(1+\frac{r-2}{k-r+1}\right)\right) + 2(k-i).\end{aligned}$$

Since

$$\sum_{i=1}^{r-1}(k-i) = (k-r/2)(r-1),$$

we have by (20) that

$$\begin{aligned}&|W_1| + \ldots + |W_{r-1}| + |T_2| + \cdots + |T_{r-1}|\\ &\qquad \le (\ell+1)\left(\frac{1}{k-r+1} + \ln\left(1+\frac{r-2}{k-r+1}\right)\right)(k-r/2)(r-1)\\ &\qquad\qquad + (2k-r)(r-1) + \ell + 1\\ &\qquad \le k(\ell+1)r\left(\frac{1}{k-r+1} + \ln\left(1+\frac{r-2}{k-r+1}\right)\right)\\ &\qquad\qquad + (2k-r)(r-1) + \ell + 1.\end{aligned}$$

As also $|S_i| \le |V(P_{\ell-1}^{(k)})| = (k-1)(\ell-1)+1$ for each $1 \le i \le r-1$ and

$$|V_r| = |V| - \sum_{i=1}^{r-1}(|W_i| + |T_i| + |S_i|),$$

we finally obtain, using the lower bound on $n = |V|$, that

$$\begin{aligned}|V_r| &\geq k(\ell+1)r - (2k-r)(r-1) - \ell - 1 - (r-1)\left[(k-1)(\ell-1)+1\right] \\ &= \ell(2r-3) + (r-1)(r-2) + (k-1) + \ell(k-r+1) \geq \ell(k-r+1),\end{aligned}$$

since the first three terms in the last line are nonnegative.

To prove the $O(n^k)$ complexity time, observe that in the worst-case scenario we need to go over all edges colored by the first $r-1$ colors and no edge is visited more than once.

References

1. Davies, E., Jenssen, M., Roberts, B.: Multicolour Ramsey numbers of paths and even cycles. Eur. J. Comb. **63**, 124–133 (2017)
2. Dudek, A., Prałat, P.: On some multicolor Ramsey properties of random graphs. SIAM J. Discrete Math. **31**(3), 2079–2092 (2017)
3. Erdős, P., Gallai, T.: On maximal paths and circuits of graphs. Acta Math. Acad. Sci. Hungar **10**, 337–356 (1959)
4. Faudree, R.J., Schelp, R.H.: Path Ramsey numbers in multicolorings. J. Comb. Theory Ser. B **19**(2), 150–160 (1975)
5. Figaj, A., Łuczak, T.: The Ramsey number for a triple of long even cycles. J. Comb. Theory Ser. B **97**(4), 584–596 (2007)
6. Gerencsér, L., Gyárfás, A.: On Ramsey-type problems. Ann. Univ. Sci. Budapest. Eötvös Sect. Math. **10**, 167–170 (1967)
7. Gyárfás, A., Ruszinkó, M., Sárközy, G., Szemerédi, E.: Three-color Ramsey numbers for paths. Combinatorica **27**(1), 35–69 (2007)
8. Gyárfás, A., Ruszinkó, M., Sárközy, G., Szemerédi, E.: Corrigendum: three-color Ramsey numbers for paths. Combinatorica **28**(4), 499–502 (2008)
9. Sárközy, G.N.: On the multi-colored Ramsey numbers of paths and even cycles. Electron. J. Comb. **23**(3), 3–53 (2016)

Cache Oblivious Sparse Matrix Multiplication

Matteo Dusefante(✉) and Riko Jacob

IT University of Copenhagen, Copenhagen, Denmark
{madu,rikj}@itu.dk

Abstract. We study the problem of sparse matrix multiplication in the Random Access Machine and in the Ideal Cache-Oblivious model. We present a simple algorithm that exploits randomization to compute the product of two sparse matrices with elements over an arbitrary field. Let $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$ be matrices with h nonzero entries in total from a field $\mathbb{F}$. In the RAM model, we are able to compute all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h + kn)$ time and $\mathcal{O}(h)$ space, where the notation $\tilde{\mathcal{O}}(\cdot)$ suppresses logarithmic factors. In the External Memory model, we are able to compute cache obliviously all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h/B + kn/B)$ I/Os and $\mathcal{O}(h)$ space. In the Parallel External Memory model, we are able to compute all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h/PB + kn/PB)$ time and $\mathcal{O}(h)$ space, which makes the analysis in the External Memory model a special case of Parallel External Memory for $P = 1$. The guarantees are given in terms of the size of the field and by bounding the size of $\mathbb{F}$ as $|\mathbb{F}| > kn\log(n^2/k)$ we guarantee an error probability of at most $1/n$ for computing the matrix product.

1 Introduction

Matrix multiplication is a fundamental operation in computer science and mathematics. Despite the effort, the computational complexity is still not settled, and it is not clear whether $\mathcal{O}(n^2)$ operations are sufficient to multiply two dense $n \times n$ matrices.

Given matrices $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$, we define h as the number of nonzero entries in the input, i.e. $h = \texttt{nnz}(A) + \texttt{nnz}(C)$, k as the number of nonzero entries in the output, i.e. $k = \texttt{nnz}(AC)$, where $\texttt{nnz}(A)$ denotes the number of nonzero entries in matrix A. Let $A_{i,j}$ be the value of the entry in the matrix A with coordinates (i, j). We denote with $A_{*,j}$ and $C_{i,*}$ the j-th column of A and the i-th row of C respectively. Note that we can easily detect, by scanning the input matrices, null vectors. Hence, without loss of generality, we consider only the case where $h \geq 2n$ and the rows of A (resp. columns of C) are not n-dimensional null vectors. Observe that, in contrast cancellations can lead to situations where $k \leq n$.[1] The algorithms presented in this paper can compute the product of matrices of arbitrary size and the bounds can be straightforwardly extended to rectangular matrices. However, for the ease of exposition, we restrict our analysis to square matrices. We denote with *column major layout* the lexicographic order where the entries of A are sorted by column first, and by row index within a column. Analogously, we denote with *row major layout* the order where the entries of A are sorted row-wise first, and column-wise within a row. Note that a row major layout can be obtained from column major layout by transposing A, and vice versa. Throughout this paper we use $f(n) = \tilde{\mathcal{O}}(g(n))$ as shorthand for $f(n) = \mathcal{O}(g(n) \log^c g(n))$ for some constant c. The memory hierarchy we refer to is modeled by the I/O model by Aggarwal and Vitter [1], the Ideal Cache-Oblivious model by Frigo et al. [2] and the Parallel External Memory model by Arge et al. [3]. We denote with M the number of elements that can fit into internal memory, B the number of elements that can be transferred in a single block and P as the number of processors. The parameters M, and B are referred as the *memory size* and *block size* respectively. The Ideal Cache-Oblivious model resembles the I/O model except for memory and block size are unknown to the algorithm. Unless otherwise stated, it holds $1 \leq B \leq M \ll h$. Note that, for our parallel algorithm, we consider a cache aware model since concurrency is nontrivial in external memory models whenever the block size B is unknown [4].

1.1 Contributions

We study the problem of matrix multiplication in the Random Access Machine and in external memory over an arbitrary field $(\mathbb{F}, +, \cdot)$, where $(+, \cdot)$ are atomic operations over elements of $\mathbb{F}$ in the computational models. We present a randomized algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ that, after $\mathcal{O}(h)$ time for preprocessing using deterministic $\mathcal{O}(h)$ space, computes, using $\mathcal{O}(nk \log(n^2/k))$ time all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability. We present a cache oblivious algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$. After $\mathcal{O}((h/B) \log_{M/B} h/B)$ I/Os for preprocessing, using deterministic $\mathcal{O}(h)$ space, we are able to compute, using $\mathcal{O}((n/B)k \log(n^2/k))$ I/Os, all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability, under a tall cache assumption, i.e. $M \geq B^{1+\varepsilon}$ for some $\varepsilon > 0$. Similarly, in the Parallel External Memory model, we present an algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ that, after $\mathcal{O}((h/PB) \log_d(h/B))$ I/Os for preprocessing, with $d = \max\{2, \min\{M/B, H/PB\}\}$, using deterministic $\mathcal{O}(h)$ space, computes, using $\mathcal{O}((n/PB + \log P)k \log(n^2/k))$ I/Os, all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability. Note that, for the External Memory model and the Parallel External Memory model, preprocessing is dominated by $\mathcal{O}(\text{sort}(h))$ I/Os which stems from layout transposition. Although faster

[1] A cancellation occurs when $(AC)_{i,j} = 0$ while elementary products do not evaluate to zero, i.e. $A_{i,\kappa} \cdot C_{\kappa,j} \neq 0$, for some $\kappa \in [n]$.

algorithms for transposing sparse matrices exist, for the ease of exposition, we consider $\mathcal{O}(\text{sort}(h))$ I/Os as an upper bound for preprocessing which weakens the bounds only in the parameters of the logarithmic factors. We give rigorous guarantees on the probability of detecting all the nonzero entries of the output matrix by studying how the process of generating random elements from the field affects the process of locating entries. The guarantees are given in terms of the size of the field. If the algorithms generate random variables from an arbitrary field $\mathbb{F}$ then we are able to detect a nonzero entry in the matrix with probability at least $1 - 2/|\mathbb{F}| + 1/|\mathbb{F}|^2$. On the other hand, given an arbitrary field $\mathbb{F}$, if the random variables are generated from $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$ then we detect a nonzero entry with probability at least $1 - 1/|\mathbb{F}^*|$. By bounding the size of $\mathbb{F}$ as $|\mathbb{F}| \geq ckn\log(n^2/k)$, for some constant c, we guarantee an error probability of at most $1/n$. Conversely, if $|\mathbb{F}| < ckn\log(n^2/k)$ we can improve the error probability by repeating the random process an arbitrary number of time, say $\log n$ times, thus sacrificing a $\log n$ factor in space and time with the effect of decreasing the error probability by a factor of n.

1.2 Related Work

Given two $n \times n$ matrices A and C, the matrix product AC can be trivially computed with $\mathcal{O}(n^3)$ arithmetic operations. Strassen [5], in 1969, provided a recursive algorithm able to multiply two matrices in $\mathcal{O}(n^\omega)$ with $\omega = 2.8073549$ by exploiting cancellations. The last known result is due to Le Gall [6] who holds the current record of $\omega < 2.3728639$. Yuster and Zwick [7] presented an algorithm that multiplies two $n \times n$ matrices over a ring using $\tilde{\mathcal{O}}(h^{0.7}n^{1.2} + n^{2+o(1)})$ arithmetic operations. Iwen and Spencer [8] proved that if each column of AC contains at most $n^{0.29462}$ nonzero entries, then A and C can be multiplied with $\mathcal{O}(n^{2+\varepsilon})$ operations. Our algorithms improve over Yuster and Zwick [7] as well as Iwen and Spencer [8] when $k < n$ and $h \ll n^2$. In addition, we do not require a balanced assumption of the output matrix, e.g. the number of nonzero entries per column, as in [8]. Amossen and Pagh [9] provided a sparse, output-sensitive matrix multiplication that incurs in $\tilde{\mathcal{O}}(h^{2/3}k^{2/3} + h^{0.862}k^{0.408})$ operations and $\tilde{\mathcal{O}}(h\sqrt{k}/(BM^{1/8}))$ I/Os. Lingas [10] presented an output-sensitive, randomized algorithm for computing the product of two $n \times n$ boolean matrices using $\tilde{\mathcal{O}}(n^2k^{\omega/2-1})$ operations. Compared to Amossen and Pagh and Lingas, we allow cancellations of terms and we do not restrict our analysis to boolean matrices. In addition, Amossen and Pagh's I/O algorithm is not cache oblivious, i.e. it requires knowledge of of the memory size. Pagh [11] presented a randomized algorithm that computes an unbiased estimator of AC in time $\tilde{\mathcal{O}}(h + nk)$, with guarantees given in terms of the Frobenius norm. Pagh's compressed algorithm achieves the same time bounds as our algorithms. However, we improve over space complexity whenever $k < h/\log n$, i.e. when cancellations account for a $1/\log n$ factor compared to the number of input entries. Besides this, Pagh's result is algorithmically more involved, since it makes use of hash functions and Fast Fourier Transform. Williams and Yu [12] provided an output-sensitive, randomized algorithm for matrix multiplication with elements over any field, that, after $\mathcal{O}(n^2)$ preprocessing, computes each nonzero entry of the matrix product in $\tilde{\mathcal{O}}(n)$ time. We extend their techniques to the sparse input case and we improve whenever $h \ll n^2$, i.e. when the input matrices are sparse, both in time and space complexity. Jacob and Stöckel [13] presented a Monte Carlo algorithm that uses $\tilde{\mathcal{O}}(n^2(k/n)^{\omega-2} + h)$ operations and, with high probability, outputs the nonzero elements of the matrix product. In addition, they state an I/O complexity of $\tilde{\mathcal{O}}(n^2(k/n)^{\omega-2}/(M^{\omega/2-1}B) + n^2/B)$. Their analysis is focused specifically on dense matrices and we improve over their results, both in time and I/O complexity, whenever k is asymptotically smaller than n in the general case while we achieve the same bounds when $k = n$. In addition, we do not require knowledge of the memory size as opposed to [13]. Van Gucht et al. [14] presented a randomized algorithm for multiplying two boolean matrices in $\tilde{\mathcal{O}}(k + h\sqrt{k} + h)$ time. In contrast to their results, our algorithms are not restricted to the boolean case and we are able to compute the product of matrices from an arbitrary field. Matrix multiplication has been widely studied in external memory as well. In a restricted setting, i.e. the semiring model, Hong and Kung [15] provided a lower bound of $\Omega(n^3/\sqrt{M})$ I/Os for multiplying two $n \times n$ matrices using n^3 operations and a memory of size M. Frigo et al. [2] provided a cache oblivious algorithm for multiplying two $n \times n$ matrices cache obliviously using $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2/B + n^3/(B\sqrt{M}))$ I/Os. In the I/O model, Pagh and Stöckel [16] provided a randomized, I/O optimal algorithm for matrix multiplication that incurs in $\tilde{\mathcal{O}}((h/B)\min\{\sqrt{k}/\sqrt{M}, h/M\})$ I/Os. However, their algorithm does not allow cancellation of terms and it requires knowledge of the memory size in order to partition the input matrices. In relation to this, we require no knowledge on the size of M and our algorithm run cache obliviously. To the knowledge of the authors, there are no previously known cache oblivious algorithms for sparse matrix multiplication that exploit cancellations.

2 Algorithms

Williams and Yu [12] provided a simple output-sensitive algorithm for matrix multiplication. The intuition behind [12] is that nonzero entries in a submatrix of AC with indices in $[i_1, i_2] \times [j_1, j_2]$ can be detected by testing whether $\langle a, c\rangle = 0$, where $a = \sum_{k=i_1}^{i_2} u_k A_{k,*}$ and $c = \sum_{k=j_1}^{j_2} v_k C_{*,k}$ are random (w.r.t u_k and v_k) linear combinations, i.e. sketches, of rows of A and columns of C respectively. A preprocessing phase, where prefix sums are involved, allows to compute sketches a and c of arbitrary size in linear time during the query process. When the input matrices are sparse, the prefix sums densify the matrices thus having to compute and store n^2 elements. In addition, sparse matrices make nontrivial to compute linear combinations since row/column vectors are not explicitly stored.

We refine the analysis of [12] as follows. In order to detect the k positions (i, j) such that $(AC)_{i,j} \neq 0$, using binary search among the n^2 feasible locations, we need at most $k \log n^2$ comparisons. We note that the algorithms do not yield false positives when querying submatrices. That is, given an all-zero submatrix with related sketches a and c, it holds $\langle a, c\rangle = 0$ always. This leads to the following lemma.

Lemma 1. *Let $\mathbb{F}$ be an arbitrary field and let $A = \{a_1, \ldots, a_n\}$ and $C = \{c_1, \ldots, c_n\}$ be two sets of d-dimensional vectors such that $a_i, c_j \in \mathbb{F}^d$, for all $i, j \in [n]$. In addition, let $u_1, \ldots, u_n, v_1, \ldots, v_n$ be $2n$ random variables chosen uniformly at random from $\mathbb{F}$ and let a, c be vectors computed as $a = \sum_{k=1}^{n} u_k a_k$, $c = \sum_{k=1}^{n} v_k c_k$. If $\langle a_i, c_j \rangle = 0$, for all $i, j \in [n]$, then $\langle a, c \rangle = 0$.*

As a consequence, at most k queries produce a positive answer, i.e. $\langle a, c \rangle \neq 0$. Via a level-by-level top-down analysis, we note that at most $\min\{2^i, 2k\}$ nodes are explored at each recursive level, with $i \in [\log n^2]$. Hence, we deduce the following.

$$\sum_{i=1}^{\log n^2} \min\{2^i, 2k\} = \sum_{i=1}^{\log k} 2^i + \sum_{i=\log k}^{\log n^2} k \leq 2k + 2k \log(n^2/k). \tag{1}$$

Accordingly, we recursively split AC into two evenly divided submatrices, which resembles the splitting phase of a k-d tree. We query each submatrix and after at most $\log(n^2/k)$ queries we isolate each nonzero entry. In the following theorem we show how to compute linear combinations of sparse matrices. The intuition is to preserve the sparseness of the input matrices while computing prefix sums and generate sketches via predecessor queries, which can be efficiently computed using *fractional cascading* [17].

Theorem 1 (RAM). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n \times n}$, $C \in \mathbb{F}^{n \times n}$ and assume A and C have h nonzero entries. After $\mathcal{O}(h)$ time for preprocessing and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n \times n}$ w.h.p, using $\mathcal{O}(kn \log(n^2/k))$ time.*

Proof. We assume that the input matrices A and C are stored in column major and row major layout respectively. If not, we can transpose A and C using $\mathcal{O}(h)$ time and $\mathcal{O}(h)$ additional space.

Preprocessing: We generate vectors $u = (u_1, \ldots, u_n) \in \mathbb{F}^n$ and $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$ uniformly at random and we initialize the data structures $\mathcal{A}$ and $\mathcal{C}$ as follows: for each $A_{i,j} \neq 0$ and $C_{i,j} \neq 0$ then

$$\mathcal{A}_{i,j} = \sum_{k=1}^{i} u_k A_{k,j} \quad \mathcal{C}_{i,j} = \sum_{k=1}^{j} v_k C_{i,k} \quad A_{k,j}, C_{i,k} \neq 0, i, j \in [n]. \tag{2}$$

Intuitively, $\mathcal{A}_{i,j}$ (resp. $\mathcal{C}_{i,j}$) denotes the prefix sum of the nonzero entries of the column vector $A_{*,j}$ up to row i (row vector $C_{i,*}$ up to column j). After this phase, $\mathcal{A}$ and $\mathcal{C}$ maintain the same sparse structure, as well as the same layout, of the original input matrices. Computing $\mathcal{A}$ and $\mathcal{C}$ requires $\mathcal{O}(h)$ time and $\mathcal{O}(h)$ space.[2] Starting from column $j = n - 1$, every column vector $\mathcal{A}_{*,j}$ is augmented with every element in even position from the sparse column vector $\mathcal{A}_{*,j+1}$. After the augmentation, the vector $\mathcal{A}_{*,j}$ contains entries native to

[2] Initializing $\mathcal{A}$ and $\mathcal{C}$ corresponds to computing prefix sums of each row and column vector of A and C respectively, which requires a linear scan of the input matrices.

$\mathcal{A}_{*,j}$ and entries inherited from $\mathcal{A}_{*,j+1}$. For each inherited entry, we add pointers to its native-predecessor and its native-successor. If $\mathcal{A}_{1,j}$ is undefined, every column vector stores a *dummy* entry in first position with value 0. For each entry in $\mathcal{A}_{*,j}$, we add a *bridge* to the entry with the same row index in $\mathcal{A}_{*,j+1}$ or, if it is undefined, we add a bridge to the predecessor. Dummy entries ensure that every element in $\mathcal{A}_{*,j}$ has at least a bridge towards $\mathcal{A}_{*,j+1}$. The augmentation, together with bridging, requires a linear scan of the column vectors. The space required by the augmented vectors is $T(j) = \mathtt{nnz}(A_{*,j}) + T(j+1)/2 + 1$, with $T(n) = \mathtt{nnz}(A_{*,n})$ and $j \in [n-1]$, which is a geometric series bounded by $2h$. The data structure $\mathcal{A}$ is further augmented with a dense vector $\mathcal{A}_{*,0}$ where every $\mathcal{A}_{i,0}$ has a bridge to either the entry with the same row index or its predecessor in $\mathcal{A}_{*,1}$. The total space required is $2h + n \leq 3h$. Analogous considerations hold for data structure $\mathcal{C}$.

Computing AC: We recursively divide AC into two evenly divided submatrices (which resembles the splitting of a k-d tree) and query each submatrix in order to detect nonzero entries. Each query is answered via an inner product $\langle a, c\rangle$ where sketches a and c are constructed using fractional cascading. Given a generic submatrix of AC with indices in $[i_1, i_2] \times [j_1, j_2]$ we compute sketches of matrix A with rows in $[i_1, i_2]$ and of matrix C with columns in $[j_1, j_2]$ respectively. We start by indexing $\mathcal{A}_{i_1,0}$ which redirects to an entry $\mathcal{A}_{i,1}$. We probe the data structure for the native-predecessor, call it $\mathcal{A}_{i_p,1}$, and the native-successor, call it $\mathcal{A}_{i_s,1}$, of $\mathcal{A}_{i,1}$. Recall $i_2 \geq i_1$ and $i \leq i_1$.

1. If $\mathcal{A}_{i,1}$ is native then: (a) if $i_s < i_1$ then we emit $\mathcal{A}_{i_s,1}$, (b) if $i = i_1$ then we emit $\mathcal{A}_{i_p,1}$, (c) otherwise we emit $\mathcal{A}_{i,1}$.
2. If $\mathcal{A}_{i,1}$ is inherited then: (a) if $i_s < i_1$ then we emit $\mathcal{A}_{i_s,1}$, (b) otherwise we emit $\mathcal{A}_{i_p,1}$.

Note that, if the predecessor or the successor of $\mathcal{A}_{i,j}$ is not defined in the j-th column vector we simply output 0 or $\mathcal{A}_{i,j}$ respectively. Accordingly, we correct the following lookup by redirecting the search from either the successor $\mathcal{A}_{i_s,1}$, if $i_s < i_1$, or to $\mathcal{A}_{i,1}$, otherwise, and following its bridge to $\mathcal{A}_{i,2}$. We iterate the process up to the n-th column and we produce a n-dimensional vector a_{i_1}. The process for i_2 is analogous. Note that, for i_2, the case (1b) is omitted and inequalities become non-strict as we want to capture the elements with row index i_2. After cascading through the n columns we have vectors a_{i_1} and a_{i_2}. The sketch of the submatrix A with row indices in $[i_1, i_2]$ stems from $a = a_{i_2} - a_{i_1}$, i.e. the element-wise difference. We repeat the same process for $\mathcal{C}$ thus computing c and we query the submatrix of AC by performing the inner product $\langle a, c\rangle$. The construction of sketches a and c requires to probe the data structure a constant number of times per column and per row respectively. Hence, $\mathcal{O}(n)$ time is required per query. By Formula (1) at most $k \log(n^2/k)$ queries are required to isolate the k nonzero entries of AC. The claim follows. □

The algorithm from Theorem 1 computes k locations (i, j) to as many nonzero entries in $AC \in \mathbb{F}^{n\times n}$. In order to compute $(AC)_{i,j}$ we can retrieve, using

Formula (2), the entry value as follows $(AC)_{i,j} = \langle A_{i,*}, C_{*,j} \rangle = \sum_k \left[(\mathcal{A}_{i,k} - \mathcal{A}_{i-1,k})(\mathcal{C}_{k,j} - \mathcal{C}_{k,j-1})\right]/u_i v_j$ while querying unit length matrices.

2.1 External Memory and Parallel External Memory

Fractional cascading relies on random memory accesses for cascading through $\mathcal{A}_{*,j}$, with $j > 1$. In the worst case, $\mathcal{O}(n)$ blocks must be loaded in memory. Instead, we use a data structure which is close in spirit to the *range coalescing* data structure by Demaine et al. [18].

Theorem 2 (Ideal Cache-Oblivious). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ and assume A and C have h nonzero entries. Let $M \geq B^{1+\varepsilon}$ for some $\varepsilon > 0$. After $\mathcal{O}((h/B)\log_{M/B}(h/B))$ I/Os for preprocessing and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n\times n}$ w.h.p., using $\mathcal{O}((kn/B)\log(n^2/k))$ I/Os.*

Proof. We describe the procedure for preprocessing matrix A and generating the sketch a. We transpose the input matrix A in column major layout using $\mathcal{O}((h/B)\log_{M/B}(h/B))$ I/Os with a cache oblivious sorting algorithm [19] (this requires the tall cache assumption $M \geq B^{1+\varepsilon}$) and we compute column-wise prefix sums using $\mathcal{O}(h/B)$ I/Os. Given the matrix A, we generate a sparse 0–1 representation A' of A, where $A'_{i,j} = 1$ if and only if $A_{i,j} \neq 0$, $A'_{i,j} = 0$ otherwise, using $\mathcal{O}(h/B)$ I/Os. We compute a counting vector $H = A'\mathbf{1}$, where $\mathbf{1} \in 1^n$ and $H_i = \sum_i \mathtt{nnz}(A_{i,*})$, using a cache oblivious Sparse Matrix Vector Multiplication algorithm [20] and $\mathcal{O}((h/B)\log_{M/B}(n/M))$ I/Os. After a prefix sum over H we are able to emit h/n index positions $r_l \in [n]$ such that $\sum_{i=r_l}^{r_{l+1}} \mathtt{nnz}(A_{i,*}) \leq 3n$. As a consequence, we build h/n buckets $\mathcal{A}_l$ of size $\mathcal{O}(n)$ where the elements of $\mathcal{A}_l$ are the entries $A_{i,j}$ such that $i \in [r_l, r_{l+1})$. Starting from $\mathcal{A}_2$, we incrementally augment the bucket $\mathcal{A}_l$ with elements from $\mathcal{A}_{l-1}$ such that, after the augmentation, for every column index j, there is an entry with value equal to the prefix sum up to bucket l. As in Theorem 1, we augment the data structure with a column vector $\mathcal{A}_{*,0}$ of size n, where $\mathcal{A}_{i,0}$ indices the l-th bucket if and only if $i \in [r_l, r_{l+1})$, with $l \in [h/n]$. A query on the data structure $\mathcal{A}$ probes $\mathcal{A}_{i_1,0}$ using a single I/O and it incurs in $\mathcal{O}(n/B)$ I/Os for scanning the bucket, thus generating the sketch a. Analogously, we generate the sketch c and we compute the inner product $\langle a, c\rangle$ by scanning the vectors using $\mathcal{O}(n/B)$ I/Os. □

Corollary 1 (Parallel External Memory). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$, assume A and C have h nonzero entries and let $P \leq n/B$. After $\mathcal{O}((h/PB)\log_d(h/B))$ I/Os for preprocessing, with $d = \max\{2, \min\{M/B, H/PB\}\}$, and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n\times n}$ w.h.p., using $\mathcal{O}((n/PB + \log P)k\log(n^2/k))$ I/Os.*

3 Probabilistic Error Analysis

We proceed to give guarantees on the probability of detecting non-zero entries in the output matrix and we study how altering the process of random generation alters the probability of detection. The guarantees are given in terms of the field size and not on the size of the matrix as, e.g., in [11]. Throughout the paper we gave no restriction on the field $\mathbb{F}$. Nevertheless, when $\mathbb{F}$ is infinite and countable, we require to sample from a finite subset of $\mathbb{F}$. This constraint is justified since random variables cannot be uniformly distributed among infinite and countable sets. Fields, in contrast with other algebraic structures, guarantee the existence of the multiplicative inverse for elements of $\mathbb{F}$, a property we use for proving the following lemmas.

Lemma 2. *Let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ and let $AC \in \mathbb{F}^{n\times n}$ have at most k nonzero entries. Consider a submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$ and assume to query the submatrix with sketches a, c as in Theorem 1. (i) The matrix has a nonzero entry if and only if $\langle a, c\rangle \neq 0$ with probability at least $1 - 2/|\mathbb{F}| + 1/|\mathbb{F}|^2$. (ii) The submatrix is all zero if and only if $\langle a, c\rangle = 0$ with probability at least $1 - 2k \log(n^2/k)/|\mathbb{F}| + k \log(n^2/k)/|\mathbb{F}|^2$.*

$\mathbf{Pr}(\langle a, c\rangle = \langle u_i a_i, v_j c_j\rangle = 0)$, with $\langle a_i, c_j\rangle \neq 0$, is given by the probability of choosing either u_i or v_j zero uniformly at random from $\mathbb{F}$. By altering the algorithm, such that random entries are now generated from $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$, we derive the following lemma.

Lemma 3. *Let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ and let $AC \in \mathbb{F}^{n\times n}$ have at most k nonzero entries. Let $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$, consider the submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$ and assume to query the submatrix with sketches a, c as in Theorem 1 where the entries of the vectors u and v are chosen uniformly at random from $\mathbb{F}^*$. (i) The submatrix has a nonzero entry if and only if $\langle a, c\rangle \neq 0$ with probability at least $1 - 1/|\mathbb{F}^*|$. (ii) The submatrix is all zero if and only if $\langle a, c\rangle = 0$ with probability at least $1 - k \log((n^2/k) - 1)/|\mathbb{F}^*|$.*

A Omitted Proofs

Proof (Lemma 2). (i) If $\langle a, c\rangle \neq 0$, then there exist $i, j \in [n]$ such that $u_i, v_j \neq 0$ and $\langle a_i, c_j\rangle \neq 0$, hence, $(AC)_{i,j} \neq 0$. If there is a nonzero entry then $\langle a, c\rangle \neq 0$ with probability at least $1-2/|\mathbb{F}|+1/|\mathbb{F}|^2$. This is equivalent of saying that if there is a nonzero entry then $\langle a, c\rangle = 0$ with probability at most $2/|\mathbb{F}|-1/|\mathbb{F}|^2$. Without loss of generality, let $i_1 = i_2 = i$ and $j_1 = j_2 = j$. Considering a bigger submatrix with exactly one nonzero entry leaves the probability unchanged, while considering more nonzero entries will only increase the probability of $\langle a, c\rangle \neq 0$. Therefore, we consider the case where we want to isolate, with high probability, the location of a single nonzero entry in a submatrix of unit size. It follows that, in order to query the submatrix we have to perform the following inner product $\langle a, c\rangle = \langle u_i a_i, v_j c_j\rangle$, where u, v are chosen uniformly at random from $\mathbb{F}$. Since $\langle a_i, c_j\rangle \neq 0$ by hypothesis, we have that $\mathbf{Pr}(\langle a, c\rangle = 0) \geq 2/|\mathbb{F}| - 1/|\mathbb{F}|^2$.

(ii) If the submatrix of AC with indices $[i_1, i_2]\times[j_1, j_2]$ is all zero then $\langle a, c\rangle = 0$ with probability at least $1 - 2k \log(n^2/k)/|\mathbb{F}| + k \log(n^2/k)/|\mathbb{F}|^2$. By Lemma 1 this is true. If $\langle a, c\rangle = 0$ then the submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$

is all zero with probability at least $1-2k\log(n^2/k)/|\mathbb{F}|+k\log(n^2/k)/|\mathbb{F}|^2$. That is, if $\langle a,c\rangle=0$ then the submatrix has a nonzero entry with probability at most $2k\log(n^2/k)/|\mathbb{F}|-k\log(n^2/k))/|\mathbb{F}|^2$. Without loss of generality, let $i_1=i_2=i$ and $j_1=j_2=j$. We have that $\langle a,c\rangle=\langle ua_i,vc_j\rangle=0$, where u, v are chosen uniformly at random from $\mathbb{F}$. Therefore, $\mathbf{Pr}(\langle a_i,c_j\rangle\neq 0)\geq 2/|\mathbb{F}|-1/|\mathbb{F}|^2$. The latter is a lower bound on the probability to not detect a nonzero entry in the output matrix. A union bound over the $k\log(n^2/k)$ queries needed to isolate the k nonzero entries, gives us the probability to incur in at least one false negative. By considering its complement, the claim follows. □

Proof (Lemma 3). (i) If $\langle a,c\rangle\neq 0$, then there exist $i,j\in[m]$ such that $u_i,v_j\neq 0$ and $\langle a_i,c_j\rangle\neq 0$, hence, $(AC)_{i,j}\neq 0$. If there is a nonzero entry then $\langle a,c\rangle\neq 0$ with probability at least $1-1/|\mathbb{F}^*|$. This is equivalent of saying that if there is a nonzero entry then $\langle a,c\rangle=0$ with probability at most $1/|\mathbb{F}^*|$. If $i_1=i_2=i$, $j_1=j_2=j$ and $\langle a_i,c_j\rangle\neq 0$ then $\langle a,c\rangle\neq 0$ since scaling vectors with random elements from $\mathbb{F}^*$ preserves non orthogonality. If $i_1<i_2$ and $j_1<j_2$ and the submatrix contains exactly one nonzero entry, the same reasoning applies. If the submatrix has $\ell>1$ nonzero entries, then, without loss of generality, there exist $a_1,\ldots,a_\ell,c_1,\ldots,c_\ell$ such that $\langle u_1a_1,v_1c_1\rangle+\cdots+\langle u_\ell a_\ell,v_\ell c_\ell\rangle=0$ and $\langle a_i,c_j\rangle\neq 0$, for all $i,j\in[\ell]$. That is, ℓ inner products that generate as many nonzero entries and produce a false negative when the submatrix is queried. By the linearity of the inner product, we have that $\langle u_1a_1,v_1c_1\rangle+\cdots+\langle u_\ell a_\ell,v_\ell c_\ell\rangle=u_1v_1\langle a_1,c_1\rangle+\cdots+u_\ell v_\ell\langle a_\ell,c_\ell\rangle$. Hence, the sum cancels whenever $u_i=-(u_1v_1\langle a_1,c_1\rangle+\cdots+u_\ell v_\ell\langle a_\ell,c_\ell\rangle)/v_i\langle a_i,c_i\rangle$ for a generic $i\in[\ell]$. Note that, such a u_i is in $\mathbb{F}$ since fields guarantee the existence of additive and multiplicative inverses. The probability to choose u_i such that it cancels the other inner products is the same as choosing an element from $\mathbb{F}^*$ uniformly at random, i.e. $1/|\mathbb{F}^*|$.

(ii) If the submatrix of AC with indices $[i_1,i_2]\times[j_1,j_2]$ is all zero then $\langle a,c\rangle=0$ with probability at least $1-k\log((n^2/k)-1)/|\mathbb{F}^*|$. By Lemma 1 this is true. If $\langle a,c\rangle=0$ then the submatrix of AC with indices $[i_1,i_2]\times[j_1,j_2]$ is all zero with probability at least $1-k\log((n^2/k)-1)/|\mathbb{F}^*|$. That is, if $\langle a,c\rangle=0$ then the submatrix of AC has a nonzero entry with probability at most $k\log((n^2/k)-1)/|\mathbb{F}^*|$. If $\langle a,c\rangle=0$ and $i_1=i_2=i$, $j_1=j_2=j$ then $\langle a_i,c_j\rangle=0$. The same reasoning applies for $i_1<i_2$, $j_1<j_2$ and exactly one nonzero entry in the submatrix. Let $\langle a,c\rangle=0$ and suppose there exist $a_1,\ldots,a_\ell,c_1,\ldots,c_\ell$ such that $\langle u_1a_1,v_1c_1\rangle+\cdots+\langle u_\ell a_\ell,v_\ell c_\ell\rangle=0$ and $\langle a_i,c_j\rangle\neq 0$, for all $i,j\in[\ell]$. Hence, as in (i), the sum cancels with probability $1/|\mathbb{F}^*|$. The latter is a lower bound on the probability to not detect a nonzero entry in the output matrix. A union bound over the $k\log((n^2/k)-1)$ queries needed to isolate the k nonzero entries, gives us the probability to incur in at least one false negative.[3] By considering its complement, the claim follows. □

[3] We do not consider the last layer, i.e. $\log(n^2/k)$, as it does not involve any stochastic process.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious Algorithms. In: 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE (1999)
3. Arge, L., Goodrich, M.T., Nelson, M., Sitchinava, N.: Fundamental parallel algorithms for private-cache chip multiprocessors. In: Proceedings of the 20th Annual Symposium on Parallelism in Algorithms and Architectures, SPAA 2008, pp. 197–206. ACM, New York (2008)
4. Bender, M.A., Fineman, J.T., Gilbert, S., Kuszmaul, B.C.: Concurrent cache-oblivious B-trees. In: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 228–237. ACM (2005)
5. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356 (1969)
6. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, pp. 296–303. ACM (2014)
7. Yuster, R., Zwick, U.: Fast sparse matrix multiplication. ACM Trans. Algorithms (TALG) **1**(1), 2–13 (2005)
8. Iwen, M.A., Spencer, C.V.: A note on compressed sensing and the complexity of matrix multiplication. Inf. Process. Lett. **109**(10), 468–471 (2009)
9. Amossen, R.R., Pagh, R.: Faster join-projects and sparse matrix multiplications. In: Proceedings of the 12th International Conference on Database Theory, ICDT 2009, pp. 121–126. ACM, New York (2009)
10. Lingas, A.: A fast output-sensitive algorithm for Boolean matrix multiplication. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 408–419. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04128-0_37
11. Pagh, R.: Compressed matrix multiplication. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 442–451. ACM (2012)
12. Williams, R., Yu, H.: Finding orthogonal vectors in discrete structures. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Philadelphia, PA, USA, pp. 1867–1877 (2014)
13. Jacob, R., Stöckel, M.: Fast output-sensitive matrix multiplication. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 766–778. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_64
14. Van Gucht, D., Williams, R., Woodruff, D.P., Zhang, Q.: The communication complexity of distributed set-joins with applications to matrix multiplication. In: Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, pp. 199–212. ACM, New York (2015)
15. Hong, J.W., Kung, H.T.: I/O complexity: the red-blue pebble game. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, STOC 1981, pp. 326–333. ACM, New York (1981)
16. Pagh, R., Stöckel, M.: The input/output complexity of sparse matrix multiplication. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 750–761. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_62
17. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica **1**(1), 133–162 (1986)

18. Demaine, E.D., Gopal, V., Hasenplaugh, W.: Cache-oblivious iterated predecessor queries via range coalescing. In: Dehne, F., Sack, J.-R., Stege, U. (eds.) WADS 2015. LNCS, vol. 9214, pp. 249–262. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21840-3_21
19. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 426–438. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45465-9_37
20. Bender, M.A., Brodal, G.S., Fagerberg, R., Jacob, R., Vicari, E.: Optimal sparse matrix dense vector multiplication in the I/O-model. Theory Comput. Syst. **47**(4), 934–962 (2010)

Don't Rock the Boat: Algorithms for Balanced Dynamic Loading and Unloading

Sándor P. Fekete[1(✉)], Sven von Höveling[1], Joseph S. B. Mitchell[2], Christian Rieck[1], Christian Scheffer[1], Arne Schmidt[1], and James R. Zuber[2]

[1] Department of Computer Science, TU Braunschweig, 38106 Braunschweig, Germany
{s.fekete,v.sven,c.rieck,c.scheffer,arne.schmidt}@tu-bs.de

[2] Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794, USA
joseph.mitchell@stonybrook.edu, zuber139@gmail.com

Abstract. We consider dynamic loading and unloading problems for heavy geometric objects. The challenge is to maintain balanced configurations at all times: minimize the maximal motion of the overall center of gravity. While this problem has been studied from an algorithmic point of view, previous work only focuses on balancing the *final* center of gravity; we give a variety of results for computing balanced loading and unloading schemes that minimize the maximal motion of the center of gravity during the entire process.

In particular, we consider the one-dimensional case and distinguish between *loading* and *unloading*. In the unloading variant, the positions of the intervals are given, and we search for an optimal unloading order of the intervals. We prove that the unloading variant is NP-complete and give a 2.7-approximation algorithm. In the loading variant, we have to compute both the positions of the intervals and their loading order. We give optimal approaches for several variants that model different loading scenarios that may arise, e.g., in the loading of a transport ship with containers.

1 Introduction

Packing a set of objects is a classic challenge that has been studied extensively, from a variety of perspectives. The basic question is: how can the objects be arranged to fit into a container? Packing problems are important for a large spectrum of practical applications, such as loading items into a storage space, or containers onto a ship. They are also closely related to problems of scheduling and sequencing, in which issues of limited space are amplified by including temporal considerations.

Due to space constraints, several technical details are omitted from this extended abstract. A full version with all proofs can be found at [7].

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 448–460, 2018.
https://doi.org/10.1007/978-3-319-77404-6_33

Packing and scheduling are closely intertwined in *loading* and *unloading* problems, where the challenge is not just to compute an acceptable *final* configuration, but also the process of *dynamically building* this configuration, such that intermediate states are both achievable and stable. This is highly relavant in the scenario of loading and unloading container ships, for which maintaining *balance* throughout the process is crucial (Fig. 1).

Balancedness of packing also plays an important role for other forms of shipping: Mongeau and Bes [14] showed that displacing the center of gravity by less than 75 cm in a long-range aircraft may cause, over a 10,000 km flight, an additional consumption of 4,000 kg of fuel.

Fig. 1. Loading and unloading container ships.

In this paper, we consider algorithmic problems of balanced loading and unloading. For unloading, this means planning an optimal sequence for removing a given set of objects, one at a time; for loading, this requires planning both position and order of the objects.

The practical constraints of loading and unloading motivate a spectrum of relevant scenarios. As ships are symmetric around their main axis, we focus on one-dimensional settings, in which the objects correspond to intervals. Containers may be of uniform size, but stackable up to a certain limited height; because sliding objects on a moving ship are major safety hazards, stability considerations may prohibit gaps between containers. On the other hand, containers of extremely different size pose particularly problematic scenarios, which is why we also provide results for sets of containers whose sizes are exponentially growing.

1.1 Our Contributions

Our results are as follows; throughout the paper, *items* are the objects that need to be loaded (also sometimes called *placed*) or unloaded, while *container* refers to the space that accomodates them. Furthermore, we assume all objects to have unit density, i.e., their weights correspond to their lengths. In most cases, items correspond to geometric intervals.

- For unloading, we show that it is NP-complete to compute an optimal sequence. More formally, given a set of placed intervals $\{I_1, \ldots, I_n\}$, it is NP-complete to compute an order $\langle I_{\pi(1)}, \ldots, I_{\pi(n)} \rangle$, in which intervals are removed one at a time, such that the maximal deviation of the gravity's center is minimized.
- We provide a corresponding polynomial-time 2.7-approximation algorithm. In particular, we give an algorithm that computes an order of the input intervals such that removing the intervals in that order results in a maximal deviation which is no larger than 2.7 times the maximal deviation induced by an optimal order.
- For loading, we give a polynomial-time algorithm for the setting in which gaps are not allowed. In particular, given a set of lengths values $\ell_1, \ldots, \ell_n \in \mathbb{R}_{>0}$, we require a sequence $\langle I_{\pi(1)}, \ldots, I_{\pi(n)} \rangle$ of pairwise disjoint intervals with $|I_{\pi(i)}| = \ell_{\pi(i)}$ for $i = 1, \ldots, n$ such that the following holds: Placing the interval $I_{\pi(i)}$ in the i-th step results in an n-stepped loading process such that the union of the loaded intervals is connected for all points in time during the loading process. Among these *connected placements*, we compute one for which the maximal deviation of the center of gravity is minimized.
- We give a polynomial-time algorithm for the case of stackable unit intervals. More formally, given an input integer $\mu \geq 1$, in the context of the previous variant, we relax the requirement that the union of the placed intervals has to be connected and additionally allow that the placed intervals may be stacked up to a height of μ in a stable manner, defined as follows. We say that layer 0 is completely *covered*. An interval I can be placed, i.e., covered, in layer $k \geq 1$ if the interval I is covered in all layers $0, \ldots, k-1$ and if I does not overlap with another interval already placed in layer k.
- We give a polynomial-time algorithm for the case of exponentially growing lengths. More formally, in the context of the previous variant, we require that all intervals are placed in layer 1 and assume that the lengths of the input

intervals' lengths are exponentially increasing, i.e., there is an $x \geq 2$ such that $x \cdot \ell_i = \ell_{i+1}$ holds for all $i \in \{1, \ldots, n-1\}$.

1.2 Related Work

Previous work on cargo loading covers a wide range of specific aspects, constraints and objectives. The general Cargo Loading Problem (CLP) asks for an optimal packing of (possibly heterogeneous) rectangular boxes into a given bin, equivalent to the Cutting Stock Problem [10]. Most of the proposed methods are heuristics based on (mixed) integer programming and have been studied both for heterogeneous or homogeneous items. Bischoff and Marriott [2] show that the performances of some heuristics may depend on the kind of cargo.

Amiouny et al. [1] consider the problem of packing a set of one-dimensional boxes of different weight and different length into a flat bin (so they are not allowed to stack these boxes), in such a way that after placing the last box, the center of gravity is as close as possible to a fixed target point. They prove strong NP-completeness by a reduction from 3-Partition and give a heuristic with a guaranteed accuracy within $\ell_{max}/2$ of a given target point, where ℓ_{max} is the largest box, w.r.t. length. A similar heuristic is given by Mathur [13].

Gehring et al. [9] consider the general CLP, for which (rectangular) items may be stacked, and place them in any possible position. They construct non-intersecting *walls*, i.e. packings made from similar items for slices of the original container, to generate the overall packing. They also show that this achieves a good final balancing of the loaded items. Mongeau and Bes [14] consider a similar variant for which the objective is to maximize the loaded weight. In addition, there may be other parameters, e.g., each item may have a different priority [22]. A mixed integer programming approach on this variant is given by Vancroonenburg et al. [23]. Limbourg et al. [11] consider the CLP based on the moment of inertia. Gehring and Bortfeldt [8] give a genetic algorithm for *stable* packings. Fasano [6] considers packing problems of three-dimensional *tetris*-like items in combination with balancing constraints. His work is done within the context of the Automated Transfer Vehicle, which was the European Space Agency's transportation system supporting the International Space Station (ISS).

Another variant is to consider multiple *drops*, for which loaded items have to be available at each drop-off point in such a way that a rearrangement of the other items is not required; see e.g. [3,4,12]. Davies and Bischoff [5] propose an approach that produces a high space utilization for even weight distribution. These scenarios often occur in container loading for trucks, for which the objective is to achieve an even weight distribution between the axles. For a state-of-the-art survey of vehicle routing with different loading constraints and a spectrum of scenarios, see Pollaris et al. [19].

In the context of distributing cargo by sea, two different kind of ships are distinguished: *Ro-Ro* and *Lo-Lo* ships. Ro-Ro (for roll on–roll off) ships carry wheeled cargo, such as cars and trucks, which are driven on and off the ship. Some approaches and problem variants such as multiple drops, additional loading, and optional cargo as well as routing and scheduling considering Ro-Ro ships are considered by Øvstebø et al. [15,16]. On the other hand, Lo-Lo (load on–load off) ships are cargo ships that are loaded and unloaded by cranes, so any feasible position can be directly reached from above.

While all this work is related to our problem, it differs by not requiring the center of gravity to be under control for each step of the loading or unloading process. A problem in which such a constraint is required and permanently checked is Compact Vector Summation (CVS), which asks for permutation to sum a number of k-dimensional vectors in a way that keeps each partial sum within a bounded k-dimensional ball. See Sevastianov [20,21] for a summary of results in CVS and its application in job scheduling. A different (and somewhat less serious) angle is considered by Paterson and Zwick [18] and Paterson et al. [17], who consider maximizing the reach beyond the edge of a table by stacking n identical, homogeneous, frictionless bricks of length 1 without toppling over, corresponding to keeping the center of gravity of subarrangements supported.

2 Preliminaries

An *item* is a unit interval $I := [m-\frac{1}{2}, m+\frac{1}{2}]$ with midpoint m. A set $\{I_1, \ldots, I_n\}$ of n items with midpoints $m_1, \ldots, m_n$ is *valid* if $m_i = m_j$ or $|m_i - m_j| \geq 1$ holds for all $i, j = 1, \ldots, n$. The *center of gravity* $C(I_1, \ldots, I_n)$ of a valid set $\{I_1, \ldots, I_n\}$ of items is defined as $\frac{1}{n}\sum_{i=1}^{n} m_i$.

For given a valid set $\{I_1, \ldots, I_n\}$ of items we seek orderings in which each item I_j is removed or placed such that the maximal deviation for all points in time $j = 1, \ldots, n$ is minimized. More formally, for $j = 1, \ldots, n$ and a permutation $\pi : j \mapsto \pi_j$, let $C_j := C\left(I_{\pi_j}, \ldots, I_{\pi_n}\right)$.

The Unloading Problem (Unload) seeks to minimize the maximal deviation during an unloading process of $I_1, \ldots, I_n$. In particular, given an input set $\{I_1, \ldots, I_n\}$ of items, we seek a permutation π such that $\max_{i,j=1,\ldots,n} |C_i - C_j|$ is minimized.

In the Loading Problem (Load) we relax the constraint that the positions of the considered items are part of the input. In particular, we seek an ordering and a set of midpoints for the containers such that the containers are disjoint and the maximal deviation for all points in time of the loading process is minimized; see Sect. 4 for a formal definition.

3 Unloading

We show that the problem Unload is NP-complete and give a polynomial-time 2.7-approximation algorithm for Unload. We first show that there is a polynomial- time reduction from the discrete version of Unload, the Discrete Unloading Problem (dUnload), to Unload; this leads to a proof that Unload is NP-complete, followed by a 2.7-approximation algorithm for Unload.

In the Discrete Unloading Problem (dUnload), we do not consider a set of items, i.e., unit intervals, but a discrete set $X := \{x_1, \ldots, x_n\}$ of points.

The center of gravity $C(X)$ of X is defined as $\frac{1}{n}\sum_{i=1}^{n} x_i$. For $j = 1, \ldots, n$ and a permutation $\pi : j \mapsto \pi_j$, let $C_j = C\left(x_{\pi_j}, \ldots, x_{\pi_n}\right)$. Again, we seek a permutation such that $\max_{i,j=1,\ldots,n} |C_i - C_j|$ is minimized.

Corollary 1. Unload *and* dUnload *are polynomial-time equivalent.*

3.1 NP-Completeness of the Discrete Case

We can establish NP-completeness of the discrete problem dUnload.

Theorem 1. dUnload *is* NP*-complete.*

The proof is based on a reduction of 3-Partition and omitted for lack of space; see the full version of this paper [7]. Because of the polynomial-time equivalance of dUnload and Unload, we conclude the following.

Corollary 2. Unload *is* NP*-complete.*

3.2 Lower Bounds and an Approximation Algorithm

When unloading a set of items, their positions are fixed, so (after reversing time) unloading is equivalent to a loading problem with predetermined positions. For easier and uniform notation throughout the paper, we use this latter description.

In order to develop and prove an approximation algorithm for dUnload, we begin by examining lower bounds on the span, $R - L$, of a minimal interval, $[L, R]$, containing the centers of gravity at all stages in an optimal solution.

Without loss of generality, we assume that the input points x_i sum to 0 (i.e., $\sum_i x_i = 0$), so that the center of gravity, C_n, of all n input points is at the origin. We let $R = \max_i C_i$ and $L = \min_i C_i$. Our first simple lemma leads to a first (fairly weak) bound on the span.

Lemma 1. *Let $(x_1, x_2, x_3, \ldots)$ be any sequence of real numbers, with $\sum_i x_i = 0$. Let $C_j = (\sum_{i=1}^{j} x_i)/j$ be the center of gravity of the first j numbers, and let $R = \max_i C_i$ and $L = \min_i C_i$. Then, $|R - L| \geq \frac{|x_i|}{i}$, for all $i = 1, 2, \ldots$.*

Due to space constraints, the proof of Lemma 1 is omitted; it can be found in the full version of this paper [7].

Corollary 3. *For any valid solution to* dUnload*, the minimal interval $[L, R]$ containing the center of gravity at every stage must have length $|R - L| \geq \frac{|u_i|}{i}$ where u_i is the input point with the i-th smallest magnitude.*

We note that the naive lower bound given by Corollary 3 can be far from tight: Consider the sequence $1, 2, 3, 4, 5, 6, 7, -7, -7, -7, -7$. In the optimal order, the first -7 is placed fourth, after $2, 1, 3$. The optimal third and fourth centers, $\{2, -\frac{1}{4}\}$ are the largest magnitude positive and negative centers seen, and show a span 2.25 times greater than the naive bound of 1. By placing the first -7

in the third position, $R \geq \frac{3}{2}$, and $L \leq -\frac{4}{3}$. By placing it fifth, $R \geq \frac{5}{2}$. Our observation was that failing to place our first -7 if the cumulative sum is > 7 would needlessly increase the span.

This generalizes to the sequence $(x_1 = 1, x_2 = 2, \ldots, x_{k-1} = k-1, x_k = -k, x_{k+1} = -(k+1), \ldots, x_N)$, with an appropriate x_N to make $\sum x_i = 0$. If we place positive weights in increasing order until $c_l \geq \frac{k}{l}$, placing $-k$ instead of a positive at position l would decrease the center of gravity well below $\frac{k}{l}$. The first negative should be placed when $\min_l \frac{l^2-l}{2} \geq k$, which is when $l \approx \sqrt{2k}$. In this example, our optimal center of gravity span is at least $\frac{k}{l} \approx \sqrt{\frac{k}{2}}$, not the 1 from the naive bound of Corollary 3.

We now describe our heuristic, $\mathcal{H}$, which leads to a provable approximation algorithm. It is convenient to relabel and reindex the input points as follows. Let $(P_1, P_2, \ldots)$ denote the positive input points, ordered (and indexed) by increasing value. Similarly, let $(N_1, N_2, \ldots)$ denote the negative input points, orders (and indexed) by increasing magnitude $|N_i|$ (i.e., ordered by decreasing value).

The heuristic $\mathcal{H}$ orders the input points as follows. The first point is simply the one closest to the origin (i.e., of smallest absolute value). Then, at each step of the algorithm, we select the next point in the order by examining three numbers: the partial sum, S, of all points placed in the sequence so far, the smallest magnitude point, α, not yet placed that has the same sign as S, and the smallest magnitude point, β, not yet placed that has the opposite sign of S. If $S + \alpha + \beta$ is of the same sign as S, then we place β next in the sequence; otherwise, if $S + \alpha + \beta$ has the opposite sign as S, then we place α next in the sequence. The intuition is that we seek to avoid the partial sum from drifting in one direction; we switch to the opposite sign sequence of input points in order to control the drift, when it becomes expedient to do so, measured by comparing the sign of S with the sign of $S+\alpha+\beta$, where α and β are the smallest magnitude points available in each of the two directions. We call the resulting ordering the $\mathcal{H}$-permutation. The $\mathcal{H}$-permutation puts the j-th largest positive point, P_j, in position π_j^+ in the order, and puts the j-th largest in magnitude negative point, N_j, in position π_j^- in the order, where

$$\pi_j^+ = j + \max_k \{k \,:\, \sum_{i=1}^{k} |N_i| \leq \sum_{l=1}^{j} P_l\} \text{ and } \pi_j^- = j + \max_k \{k \,:\, \sum_{i=1}^{k} P_i < \sum_{l=1}^{j} |N_l|\}.$$

We obtain an improved lower bound based on our heuristic, $\mathcal{H}$, which orders the input points according to the $\mathcal{H}$-permutation.

Lemma 2. *A lower bound on the optimal span of* dUnload *is given by* $|R - L| \geq \frac{P_i}{\pi_i^+}$ *and* $|R - L| \geq \frac{|N_i|}{\pi_i^-}$.

To prove the lemma, we begin with a claim.

Claim. For any input set to the discrete unloading problem, where s_i are all terms with the same sign sorted by magnitude, a permutation π that minimizes the maximum value of the ratio $\frac{|s_i|}{\pi_i}$ must satisfy $\pi_k < \pi_i$, for all $k < i$.

Proof. By contradiction, assume that the minimizing permutation π has the maximum value of the ratio $\frac{|s_i|}{\pi_i}$ occur at an i for which there exists a $k < i$ for which $\pi_i \leq \pi_k$, which means that $\pi_i < \pi_k$ (because π_i cannot equal π_k for a permutation π, and $k \neq i$).

Because the terms s_i are indexed in order sorted by magnitude, $|s_k| \leq |s_i|$. Exchanging the order of s_i and s_k in the permutation would lead to two new ratios in our sequence: $\frac{|s_i|}{\pi_k}$ and $\frac{|s_k|}{\pi_i}$. Because $\pi_k > \pi_i$, we get $\frac{|s_i|}{\pi_k} < \frac{|s_i|}{\pi_i}$. Because $|s_k| \leq |s_i|$, we get $\frac{|s_k|}{\pi_i} \leq \frac{|s_i|}{\pi_i}$. Because these new ratios are smaller than $\frac{|s_i|}{\pi_i}$, we get a contradiction to the fact that π minimizes the maximum ratio.

The following claim is an immediate consequence of Lemma 1.

Claim. For the i maximizing $\frac{P_i}{\pi_i^+}$, any ordering placing this element earlier than π_i^+ in the sequence has a span $|R - L| > \frac{P_i}{\pi_i^+}$. Similarly, for the i maximizing $\frac{|N_i|}{\pi_i^+}$, any ordering placing this element earlier than π_i^- in the sequence has a span $|R - L| > \frac{|N_i|}{\pi_i^-}$.

On the other hand, the following holds.

Claim. For the i maximizing $\frac{P_i}{\pi_i^+}$, any ordering placing this element later than π_i^+ in the sequence has a span $|R-L| > \frac{P_i}{\pi_i^+}$. A similar statement holds for $\frac{|N_i|}{\pi_i^-}$.

Proof. The proof is by contradiction. The index into the $\mathcal{H}$ permutation maximizing the ratio $\frac{|x_k|}{k}$ is i. We assume (wlog) $x_i = P_J > 0$, and we let $K = i - J$.

If P_J is not placed in position i, we suppose another element, x, can be placed in its stead and results in a span that is less than $\frac{P_J}{i}$.

When placing any positive $x > P_J$ in the initial i position, the lowest possible observed span from Lemma 1 is at least $\frac{x}{i} > \frac{P_J}{i}$, which would contradict our assumption. Similarly, all positive points placed before or at position i must be less than or equal to P_J.

All permutations of these $J - 1$ positive elements and the first $K + 1$ negative elements have a large negative center of gravity at position i. From $K = \max_k\{k : \sum_{i=1}^{k} |N_i| \leq \sum_{l=1}^{J} P_l\}$, we get $\sum_{i=1}^{K+1} |N_i| \geq \sum_{l=1}^{J} P_l$, and hence $\sum_{i=1}^{K+1} N_i + \sum_{l=1}^{J} P_l \leq 0$, implying $\sum_{i=1}^{K+1} N_i + \sum_{l=1}^{J-1} P_l \leq -P_J$. Therefore, the maximizing value satisfies

$$|c^*| = \frac{|\sum_{i=1}^{K+1} N_i + \sum_{l=1}^{J-1} P_l|}{i} \geq \frac{P_J}{i}$$

Because the center of gravity is at a location greater than the $\mathcal{H}$-bound, and $R \geq 0 \geq L$, this span is also greater than the $\mathcal{H}$-bound and we can neither place an element greater than P_J nor one less than P_J in place of P_J while lowering the span beneath the $\mathcal{H}$-bound.

Theorem 2. *The $\mathcal{H}$-permutation minimizes the maximum (over i) value of the ratio $\frac{|x_i|}{\pi_i}$, and thus yields a lower bound on $|R - L|$.*

For the worst-case ratio, we get the following.

Theorem 3. *The $\mathcal{H}$ heuristic yields an ordering having span $R - L$ at most 2.7 times larger than the $\mathcal{H}$-lower bound.*

Due to space constraints, the proof of Theorem 3 is omitted; it can be found in the full version of this paper [7].

Corollary 4. *There is a polynomial-time* 2.7*-approximation algorithm for* UNLOAD*.*

4 Loading

We proceed to loading problems, which requires a wider range of definitions: The positions of the objects are part of the optimization and for some loading variants, the items may have different lengths. Consider the following more general definitions:

An *item* is given by a real number ℓ. By assigning a *position* $m \in \mathbb{R}$ to an item, we obtain an interval I with length ℓ and midpoint m. For $n \geq 1$, we consider a set $\{\ell_1, \ldots, \ell_n\}$ of n items and assume $\ell_1 \geq \cdots \geq \ell_n$. Furthermore, $\{\ell_1, \ldots, \ell_n\}$ is *uniform* if $\ell := \ell_1 = \ldots = \ell_n$.

A *state* is a set $\{(I_1, h_1), \ldots, (I_n, h_n)\}$ of pairs, each one consisting of an interval I_j and an integer $h_j \geq 1$, the *layer* in which I_j lies. A state satisfies the following: (1) Two different intervals that lie in the same layer do not overlap and (2) for $j = 2, \ldots, n$, an interval in layer j is a subset of the union of the intervals in layer $j - 1$.

A state $\{(I_1, h_1), \ldots, (I_n, h_n)\}$ is *plane* if all intervals lie in the first layer.

To simplify the following notations, we denote for $j = \{1, \ldots, n\}$ the midpoint of the interval I_j by m_j. The *center of gravity* $C(s)$ of a state $s = \{(I_1, h_1), \ldots, (I_n, h_n)\}$ is defined as $\frac{1}{M}\sum_{j=1}^{n} \ell_j m_j$, where M is defined as $\sum_{j=1}^{n} \ell_j$.

A *placement* p of an n-system S is a sequence $\langle I_1, \ldots, I_n \rangle$ such that $\{(I_1, h_1), \ldots, (I_j, h_j)\}$ is a state, the *j-th state* s_j, for each $j = 1, \ldots, n$. The 0-th state s_0 is defined as $\varnothing$ and its center of gravity $C(s_0)$ is defined as 0.

Definition 1. *The* LOADING PROBLEM *(*LOAD*) is defined as follows: Given a set of n items, we are searching for a placement p such that the $n+1$ centers of gravity of the $n+1$ states of p lie close to* 0*. In particular, the* deviation $\Delta(p)$ *of a placement p is defined as* $\max_{j=0,\ldots,n} |C(s_j)|$*. We seek a placement of S with minimal deviation among all possible placements for S.*

We say that stacking is not allowed *if we require that all intervals are placed in layer* 1*. Otherwise, we say that* stacking is allowed*. For a given integer $\mu \geq 1$ we say that μ is the* maximal stackable height *if we require that all used layers are no larger than μ.*

Note that in the loading case, minimizing the deviation is equivalent to minimizing the diameter, i.e., minimizing the maximal distance between the smallest and largest extent of the centers.

4.1 Optimally Loading of Unit Items with Stacking

Now we consider the case of unit items for which stacking is allowed. We give an algorithm that optimally loads a set of unit items with stacking.

Theorem 4. *There is a polynomial-time algorithm for loading a set of unit items so that the deviation of the center of gravity is in* $[0, \frac{1}{1+\mu}]$*, where* μ *is the maximum stackable height.*

Proof. Let m_i be the midpoint of item ℓ_i. Because we are allowed to stack items up to height μ, the strategy is the following: set $m_1 = m_2 = \cdots = m_\mu = \frac{1}{1+\mu}$, i.e., the first μ items are placed at the very same position. Call these first μ items the *starting stack* $\mathcal{S}_0$. Subsequently, we place the following items on alternating sides of $\mathcal{S}_0$, i.e., the item $\ell_{\mu+1}$ is placed as close as possible on the left side of $\mathcal{S}_0$, $\ell_{\mu+2}$ is placed as close as possible on the right side, $\ell_{\mu+3}$ is placed on top of $\ell_{\mu+1}$ (if we did not already reach the maximum stackable height of μ), or next to $\ell_{\mu+1}$ (if $\ell_{\mu+1}$ is on the μ-th layer), etc.

After each placement of $\ell_i, 1 \leq i \leq \mu$, we have $C(\ell_i) = \frac{1}{1+\mu}$. After two more placed items, the center of gravity is again at $\frac{1}{1+\mu}$, because these items neutralize each other. Thus, the critical part is a placement on the left side of $\mathcal{S}_0$. We proceed to show that after placing an item on the left side, the center of gravity is at position at least 0.

The midpoint $m_{\mu+1}$ of the item $\ell_{\mu+1}$ is $\frac{-\mu}{1+\mu}$, thus $C(\ell_{\mu+1}) = \frac{\mu}{1+\mu} - \frac{\mu}{1+\mu} = 0$. Now assume that we have already placed $c = (2k+1)\cdot\mu+\zeta$ items, where $\zeta < 2\mu$ and odd, i.e., we have already placed the starting stack $\mathcal{S}_0$ and k additional stacks of height μ on each side of $\mathcal{S}_0$. Let $z := (2k+1)\cdot\mu$. Then the center of gravity is at position $C(c)$, where

$$\begin{aligned}C(c) =& \frac{z\cdot\frac{1}{1+\mu} + \sum\limits_{i=z+1}^{z+\zeta} m_i}{z+\zeta} = \frac{(z+\zeta-1)\cdot\frac{1}{1+\mu} + \frac{-k\mu-k-\mu}{1+\mu}}{z+\zeta} = \frac{k\mu+\zeta-1-k}{(1+\mu)\cdot(z+\zeta)}\\ =& \frac{k(\mu-1)+\zeta-1}{(1+\mu)\cdot(z+\zeta)} \geq \frac{\zeta-1}{(1+\mu)\cdot(z+\zeta)} \geq \frac{0}{(1+\mu)\cdot(z+\zeta)} \geq 0.\end{aligned}$$

In the following we show that there is no strategy that can guarantee a smaller deviation of the center of gravity than the strategy described in the last theorem.

Theorem 5. *The strategy given in Theorem 4 is optimal for* $n > \mu$*, i.e., there is no strategy such that the center of gravity deviates in* $[0, \frac{1}{1+\mu})$*.*

Proof. Because $n > \mu$, we must use at least two stacks. Now assume that we first place k items on one stack $\mathcal{S}_0$, before we start another one. Without loss of generality, we place this first k items at position $\frac{1}{1+\mu} - \varepsilon$. We proceed to show that for any $\varepsilon > 0$, we need k to be at least $\mu+1$, to get the new center of gravity to position $> -\varepsilon$ and therefore a smaller deviation as the strategy in Theorem 4.

If we place the item ℓ_{k+1} on the right side of $\mathcal{S}_0$, the new center of gravity gets to a position larger than $\frac{1}{1+\mu} - \varepsilon$, a contradiction. Thus, it must be placed on the left of $\mathcal{S}_0$. The position of this item has to be $-\frac{\mu}{1+\mu} - \varepsilon$. This yields the new center of gravity of $(k \cdot (\frac{1}{1+\mu} - \varepsilon) - \frac{\mu}{1+\mu} - \varepsilon)/k+1$. This center of gravity must be located to the right of $-\varepsilon$. Thus, we have

$$k \cdot (\frac{1}{1+\mu} - \varepsilon) - \frac{\mu}{1+\mu} - \varepsilon + (k+1) \cdot \varepsilon > 0 \quad \Leftrightarrow \quad k - \mu > 0 \quad \Leftrightarrow \quad k > \mu$$

Because we cannot stack $\mu + 1$ items, we cannot have any strategy achieving a deviation of $[0, \frac{1}{1+\mu} - \varepsilon]$. We conclude that our strategy given in Theorem 4 must be optimal.

Corollary 5. *With the given strategy for a uniform system where each item has length ℓ, the center of gravity deviates in $[0, \frac{\ell}{1+\mu}]$, which is optimal.*

4.2 Optimally Loading Without Stacking but With Minimal Space

Assume that the height of the ship to be loaded does not allow stacking items. This makes it necessary to ensure that the space consumption of the packing is minimal. We restrict ourselves to plane placements such that each state is connected. For simplicity, we assume w.l.o.g. that $\ell_1 \geq \cdots \geq \ell_n$ holds. First we argue that $\Delta(p) \geq \frac{\ell_2}{4}$ holds for an arbitrary connected plane placement p of S. Subsequently we give an algorithm that realizes this lower bound.

A fundamental key for this subcase is that the center of gravity of a connected plane state is the midpoint of the induced overall interval.

Observation 1. *Let s be a plane state such that the union of the corresponding intervals is an interval $[a, b] \subset \mathbb{R}$. Then $C(s) = \frac{a+b}{2}$.*

Lemma 3. *For each plane placement p of S, we have $\Delta(p) \geq \frac{\ell_2}{4}$.*

Proof. Let p be an arbitrary plane placement of $S = \langle (I_1, 1), \ldots, (I_n, 1) \rangle$, let $\langle s_0, s_1, \ldots, s_n \rangle$ be the sequence of states that are induced by p, and let $i, j \in \{1, \ldots, n\}$ be such that $I_i = |\ell_1|$ and $I_j = |\ell_1|$ hold. Observation 1 implies that $|C(s_{i-1}) - C(s_i)| = \frac{\ell_1}{4} \geq \frac{\ell_2}{4}$ and $|C(s_{j-1}) - C(s_j)| = \frac{\ell_2}{4}$. Let m_i and m_j be the midpoints of I_i and I_j. As the intervals I_i and I_j do not overlap, we conclude that $|m_i| \geq \frac{\ell_2}{2}$ or $|m_j| \geq \frac{\ell_2}{2}$ holds. W.l.o.g. assume that $|m_i| \geq \frac{\ell_2}{2}$ holds. This implies that $|C(s_{i-1})| \geq \frac{\ell_2}{4}$ or $|C(s_i)| \geq \frac{\ell_2}{4}$ holds. In both cases, we obtain $\Delta(p) \geq \frac{\ell_2}{4}$, concluding the proof.

Lemma 4. *We can compute a placement p of S such that $\Delta(p) \leq \frac{\ell_2}{4}$.*

Proof. The main idea is as follows. We remember $\ell_1 \geq \cdots \geq \ell_n$ and place the items in that order. In particular, we choose the positions of the items such that $C(s_1) := -\frac{\ell_2}{4}$ and $C(s_2) := \frac{\ell_2}{4}$. The remaining intervals are placed alternating, adjacent to the left and to the right side of the previously placed intervals.

In order to show that $C(s_i) \in [-\frac{\ell_2}{4}, \frac{\ell_2}{4}]$ holds for all $i \in \{0, \ldots, n\}$, we prove by induction that $C(s_i) \in [C(s_{i-2}), C(s_{i-1})]$ holds for all odd $i \geq 3$ and $C(s_i) \in [C(s_{i-1}), C(s_{i-2})]$ for all even $i \geq 4$. As Observation 1 implies $C(s_1) = -\frac{\ell_2}{4}$ and $C(s_2) = \frac{\ell_2}{4}$, this concludes the proof.

Let $i \geq 3$ be odd. We have $|C(s_{i-2}) - C(s_{i-1})| = \frac{\ell_{i-1}}{2}$. This is lower bounded by $\frac{\ell_i}{2}$ because $\ell_i \leq \ell_{i-1}$. Furthermore, we know that $|C(s_{i-1}) - C(s_i)| = \frac{\ell_i}{2}$. This implies $C(s_i) \in [C(s_{i-2}), C(s_{i-1})]$. The argument for the case of even $i \geq 4$ is analogous.

The combination of Lemmas 3 and 4 implies that our approach for connected placements is optimal.

Corollary 6. *Given an arbitrary system, there is a polynomial-time algorithm for optimally loading a general set of items without stacking and under the constraint of minimal space consumption for all intermediate stages.*

4.3 Optimally Loading Exponentially Growing Items

Similar to the previous section, we consider plane placements. Now we consider the case in which the items have exponentially rising lengths. This case highlights the challenges of uneven lengths, in particular when the sizes are growing very rapidly; without special care, this can easily lead to strong deviation during the loading process. We show how the deviation can be minimized.

Theorem 6. *There is a polynomial-time algorithm for optimally loading a set of items with lengths growing exponentially by a factor $x \geq 2$ in increasing order w.r.t. to their lengths.*

Details can be found in the full version of this paper [7].

5 Conclusion

We have introduced a new family of problems that aim for balancing objects w.r.t. their center of gravity during loading and unloading these objects, and have provided hardness results and optimal or constant-factor approximation algorithms.

There are various related challenges. These include sequencing problems with multiple loading and unloading stops (which arise in vehicle routing or tour planning for container ships); variants in which items can be shifted in a continuous fashion; batch scenarios in which multiple items are loaded or unloaded at once (making it possible to maintain better balance, but also increasing the space of possible choices); and higher-dimensional variants, possibly with inhomogeneous space constraints. All these are left for future work.

Acknowledgements. We would like to thank anonymous reviewers for providing helpful comments and suggestions improving the presentation of this paper. J. Mitchell is partially supported by the National Science Foundation (CCF-1526406).

References

1. Amiouny, S.V., Bartholdi, J.J., Vate, J.H.V., Zhang, J.: Balanced loading. Oper. Res. **40**(2), 238–246 (1992)
2. Bischoff, E.E., Marriott, M.D.: A comparative evaluation of heuristics for container loading. Eur. J. Oper. Res. **44**(2), 267–276 (1990)
3. Bischoff, E.E., Ratcliff, M.: Issues in the development of approaches to container loading. Omega **23**(4), 377–390 (1995)
4. Christensen, S.G., Rousøe, D.M.: Container loading with multi-drop constraints. Int. Trans. Oper. Res. **16**(6), 727–743 (2009)
5. Davies, A.P., Bischoff, E.E.: Weight distribution considerations in container loading. Eur. J. Oper. Res. **114**(3), 509–527 (1999)
6. Fasano, G.: A MIP approach for some practical packing problems: balancing constraints and tetris-like items. 4OR **2**(2), 161–174 (2004)
7. Fekete, S.P., von Höveling, S., Mitchell, J.S.B., Rieck, C., Scheffer, C., Schmidt, A., Zuber, J.R.: Don't rock the boat: algorithms for balanced dynamic loading and unloading. CoRR, abs/1712.06498 (2017). http://arxiv.org/abs/1712.06498
8. Gehring, H., Bortfeldt, A.: A genetic algorithm for solving the container loading problem. Int. Trans. Oper. Res. **4**(5–6), 401–418 (1997)
9. Gehring, H., Menschner, K., Meyer, M.: A computer-based heuristic for packing pooled shipment containers. Eur. J. Oper. Res. **44**(2), 277–288 (1990)
10. Gilmore, P., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. Oper. Res. **13**(1), 94–120 (1965)
11. Limbourg, S., Schyns, M., Laporte, G.: Automatic aircraft cargo load planning. JORS **63**(9), 1271–1283 (2012)
12. Lurkin, V., Schyns, M.: The airline container loading problem with pickup and delivery. Eur. J. Oper. Res. **244**(3), 955–965 (2015)
13. Mathur, K.: An integer-programming-based heuristic for the balanced loading problem. Oper. Res. Lett. **22**(1), 19–25 (1998)
14. Mongeau, M., Bes, C.: Optimization of aircraft container loading. IEEE Trans. Aerosp. Electron. Syst. **39**(1), 140–150 (2003)
15. Øvstebø, B.O., Hvattum, L.M., Fagerholt, K.: Optimization of stowage plans for roro ships. Comput. Oper. Res. **38**(10), 1425–1434 (2011)
16. Øvstebø, B.O., Hvattum, L.M., Fagerholt, K.: Routing and scheduling of roro ships with stowage constraints. Transp. Res. Part C: Emerg. Technol. **19**(6), 1225–1242 (2011)
17. Paterson, M., Peres, Y., Thorup, M., Winkler, P., Zwick, U.: Maximum overhang. Am. Math. Mon. **116**(9), 763–787 (2009)
18. Paterson, M., Zwick, U.: Overhang. Am. Math. Mon. **116**(1), 19–44 (2009)
19. Pollaris, H., Braekers, K., Caris, A., Janssens, G.K., Limbourg, S.: Vehicle routing problems with loading constraints: state-of-the-art and future directions. OR Spectr. **37**(2), 297–330 (2015)
20. Sevastianov, S.: On some geometric methods in scheduling theory: a survey. Discret. Appl. Math. **55**(1), 59–82 (1994)
21. Sevastianov, S.: Nonstrict vector summationin multi-operation scheduling. Ann. Oper. Res. **83**, 179–212 (1998)
22. Souffriau, W., Demeester, P., Berghe, G.V., De Causmaecker, P.: The aircraft weight and balance problem. Proc. ORBEL **22**, 44–45 (1992)
23. Vancroonenburg, W., Verstichel, J., Tavernier, K., Berghe, G.V.: Automatic air cargo selection and weight balancing: a mixed integer programming approach. Transp. Res. Part E: Logist. Transp. Rev. **65**, 70–83 (2014)

Probabilistic Analysis of Online (Class-Constrained) Bin Packing and Bin Covering

Carsten Fischer(✉) and Heiko Röglin

Department of Computer Science, University of Bonn, Bonn, Germany
carsten.fischer@uni-bonn.de, roeglin@cs.uni-bonn.de

Abstract. We study online algorithms for bin packing and bin covering in two different probabilistic settings in which the item sizes are drawn randomly or the items are adversarial but arrive in random order. We prove several results on the expected performance of well-known online algorithms in these settings. In particular, we prove that the simple greedy algorithm Dual Next-Fit for bin covering performs in the random-order setting strictly better than in the worst case, proving a conjecture by Christ et al. (Theoret Comput Sci 556:71–84, 2014).

Additionally we also study class-constrained bin packing and bin covering. In these problems, each item has not only a size but also a color and there are constraints on the number of different colors in each bin. These problems have been studied before in the classical worst-case model and we provide the first probabilistic analysis of these problems. We prove for several simple online algorithms bounds on their expected performance in the two probabilistic models discussed above. We observe that in the case of class constrained bin packing for several algorithms their performance differs with respect to the two probabilistic performance measures.

1 Introduction

Bin packing and bin covering are classical optimization problems, which have been studied extensively both as offline and online problems. In these problems, the input consists of a set of n items with sizes $s_1, \dots, s_n \in [0, 1]$ and one seeks for a partition of the items into bins. In the *bin packing problem* the goal is to partition the items into as few bins as possible such that each bin contains items with a total size of at most 1, whereas in the *bin covering problem* the goal is to partition the items into as many bins as possible such that each bin contains items with a total size of at least 1.

In addition to these pure versions, also several variations of bin packing and bin covering with additional constraints are of interest. One particular line of research is concerned with *class constrained* versions in which an additional

This research was supported by ERC Starting Grant 306465 (BeyondWorstCase).

parameter k is given and each item i has not only a size $s_i \in [0, 1]$ but also a color $c_i \in \mathbb{N}$. In the *class constrained bin packing problem* the goal is to partition the items into as few bins as possible such that each bin contains items with a total size of at most 1 and of at most k different colors. In the *class constrained bin covering problem* the goal is to find a partition into as many bins as possible such that each bin contains items with a total size of at least 1 and of at least k different colors.

The class constrained bin packing problem has been introduced in [13] and studied in a sequence of papers [7,14,16]. Its theoretical importance stems from the fact that it generalizes the classical bin packing problem and the *cardinality constrained bin packing problem* (see e.g. [1,3,11]). In cardinality constrained bin packing there is a parameter $k \in \mathbb{N}$ given and a bin must contain at most k items. From a practical point of view there are applications in production planning and video-on-demand systems [16]. Given the class constrained bin packing problem, it is natural to also study the class constrained bin covering problem, which has been introduced by Epstein et al. [6] with applications in fault-tolerant communication networks.

In this article, we focus on the online setting, in which the items arrive one after another and an algorithm has to assign each item immediately and irrevocably upon its arrival to one of the bins without knowing the items that come afterwards. We are particularly interested in probabilistic performance measures. We study the setting where the items are drawn independently and identically distributed (i.i.d.) from an adversarial distribution and the random-order model, in which an adversary chooses the set of items, but the items arrive in random order.

Up to now, it is not fully understood when the performance of algorithms coincide or differs in these two probabilistic settings. We prove several new upper and lower bounds on the competitive ratio of online algorithms in these probabilistic models both for the classical and the class constrained versions of bin packing and bin covering. For special cases we observe that even heuristics behave asymptotically optimal on random input. In the case of class constrained bin packing we observe different behaviors of the considered algorithms w.r.t. the two performance measures. Our analysis sheds new light on the nature of these two probabilistic performance measures in the context of bin packing and bin covering and its variants.

1.1 Probabilistic Performance Measures

In all problems considered in this article, an instance I is given by a sequence $(a_1, \ldots, a_n)$ of items, and we assume that the items arrive in the order specified by their indices. We denote by $\mathrm{OPT}(I)$ the value of the optimal offline solution, i.e., the minimum number of bins needed to pack the items in the (class constrained) bin packing problem and the maximum number of bins that can be covered by the items in the (class constrained) bin covering problem. Similarly, for an algorithm A we denote by $A(I)$ the value of the solution computed by A on instance I. Furthermore, we denote by $|I|$ the number of items in instance I.

The usual performance measure for an online algorithm A is its (asymptotic) competitive ratio, which essentially measures by which factor the solutions computed by A can be worse than the optimal offline solution. Since competitive analysis is based on the worst-case behavior of algorithms, it often yields too pessimistic results and, in many cases, it is not fine-grained enough to differentiate meaningfully between different algorithms. Worst-case analysis can be viewed as a game between the algorithm designer and an adversary whose goal is to select an input on which the designed algorithm performs as poorly as possible. For the reasons discussed above, we weaken the adversary by studying inputs that are to some extent random.

We first describe the probabilistic measures in terms of (class constrained) bin packing and discuss later how they can be adapted to (class constrained) bin covering. The first probabilistic model we consider is i.i.d. sampling. Let $\mathcal{I}$ denote a (possibly infinite) multiset of items and $p : \mathcal{I} \to [0,1]$ a probability measure on the set of items. Observe that in the case of class constrained bin packing an item is a tuple (s, c) consisting of a size s and a color c whereas in the normal bin packing problem p is simply a probability measure on item sizes. We will often denote the pair $(\mathcal{I}, p)$ by F. Then, I_n^F denotes a random instance $(A_1, \ldots, A_n)$, where n items are drawn independently according to F. The *asymptotic average performance ratio* of an algorithm A is defined as

$$\mathrm{AAPR}(A) = \sup_F \limsup_{n\to\infty} \mathbb{E}\left[\frac{A(I_n^F)}{\mathrm{OPT}(I_n^F)}\right].$$

We prove that for the distributions and algorithms we consider, the asymptotic average performance ratio can usually also be expressed as

$$\sup_F \limsup_{n\to\infty} \frac{\mathbb{E}\left[A(I_n^F)\right]}{\mathbb{E}\left[\mathrm{OPT}(I_n^F)\right]}.$$

In the case of (class constrained) bin covering we have to replace $\sup\limsup$ by $\inf\liminf$.

Often, we can reduce the analysis of the asymptotic average performance ratio to a restricted class $\mathcal{P}$ of distributions, the so-called *perfect-packing distributions.* We say that F is a perfect-packing distribution if we can represent F in the following way: We assume that there are $m \in \mathbb{N}$ bins that are perfectly packed in the sense that the total size of the items in each bin is exactly 1 and the number of colors in each bin is at most k or at least k for class constrained bin packing or class constrained bin covering, respectively. We denote by ℓ the total number of items and the items are numbered consecutively from 1 to ℓ. The distribution F is obtained by drawing an index i uniformly at random from $\{1, \ldots, \ell\} =: [\ell]$ and choosing the item with the corresponding index. Analogously, a *perfect-packing instance* in the random-order model is an instance in which in the optimal solution all bins are perfectly packed in the above sense.

Now we introduce the second performance measure. For an instance I, we denote by I^σ a random instance, where the items in I are randomly permuted.

Let A be an algorithm for the (class constrained) bin packing problem. Then the *asymptotic random-order ratio* $\mathrm{RR}(A)$ of A is defined as

$$\mathrm{RR}(A) = \limsup_{\mathrm{OPT}(I)\to\infty} \frac{\mathbb{E}\left[A(I^\sigma)\right]}{\mathrm{OPT}(I)}.$$

In the case of (class constrained) bin covering we have to replace lim sup by lim inf.

For all considered problems, the asymptotic average performance ratio cannot be worse than the random-order ratio, i.e., for any algorithm A we have $1 \leq \mathrm{AAPR}(A) \leq \mathrm{RR}(A)$ and $1 \geq \mathrm{AAPR}(A) \geq \mathrm{RR}(A)$ for the (class constrained) bin packing problem and the (class constrained) bin covering covering problem, respectively (see [8] or the full version of the paper).

We also study the special cases of class constrained bin packing and covering, where we have *unit sized items*. In this special case, we are given a parameter $B \in \mathbb{N}$, and all items have size $1/B$. For convenience, we will scale the item sizes to 1 and the bin capacity to B in this case.

1.2 Related Work

There is vast body of literature on the classical versions of bin packing and bin covering. We discuss only the results that are most relevant for our article. Kenyon [10] introduced the notion of asymptotic random-order ratio for bin packing and proved that the asymptotic random-order ratio of the best-fit algorithm (BF) lies between 1.08 and 1.5, while its (worst-case) competitive ratio is well-known to be 1.7 [5,15]. In contrast to this, Coffman et al. [9] showed that the random-order ratio of the next-fit algorithm (NF) equals its (worst-case) competitive ratio 2. Christ et al. [4] adapted the asymptotic random-order ratio to bin covering and proved that the random-order ratio of the dual next-fit algorithm (DNF) is at most 0.8, which was later improved to 2/3 [8]. In [8], we proved that the asymptotic average performance ratio of DNF is $0.5 + \varepsilon$ for a small constant $\varepsilon > 0$ for every discrete distribution F. However, this lower bound does not carry over to the random-order ratio of DNF and no lower bound except for the trivial bound of 0.5 is known for this.

The class constrained bin packing problem has been introduced in [13] and studied in a sequence of papers [7,14,16]. All results so far concern the competitive ratio in the classical worst-case model. In the case of unit sized items there is a lower bound of 2 for the asymptotic competitive ratio that can be achieved and this bound is achieved by the first-fit algorithm (FF) and the algorithm CS [NF] (also called ColorSets) introduced in [14]. In the general case, for all values of k, there exists an online algorithm for class constrained bin packing with a competitive ratio of at most 2.63492 [7]. Also the competitive ratios of several other online algorithms have been analyzed [7,16] and approximation schemes for the offline problem have been obtained [7].

The class constrained bin covering problem has been introduced by Epstein et al. [6]. Also this problem has not been studied in a probabilistic setting before.

Epstein et al. consider only the case of unit sized items and they prove several results. They obtain a polynomial-time algorithm for the offline problem and an upper bound of $\left(\frac{(B-1)(B-k+1)}{B(B-k)+B-1} \cdot (\frac{B-k}{B-1} + H_{k-1})\right)^{-1}$ for the competitive ratio of any online algorithm. Furthermore, they prove that DNF is not competitive for class constrained bin covering and they introduce the algorithm Color&Size and prove that it is $\Omega(1/k)$-competitive. They also introduce the algorithm FF2 and prove that its competitive ratio is exactly $1/B$.

1.3 Our Contributions

All mentioned algorithms are described in detail in Sect. 1.4.

Classical Bin Packing and Covering. We prove that for bin covering the simple greedy algorithm DNF achieves a random-order ratio of at least 0.501. While this is only a small improvement over the trivial bound of 0.5, it is the first bound that shows that DNF performs better in the random-order setting than in the worst case. This has already been conjectured in [4] and posed as an open problem. The conclusions in [4] also discuss the challenges in proving such a result. While the different bins covered by DNF are not independent in the random-order model, one main observation in our proof is that they are identically distributed. Given this observation, the proof relies on analyzing the expected overshoot of the first filled bin, where the overshoot is defined as the total size of the items assigned to that bin minus its capacity 1. We show that the expected overshoot is strictly less than 1. This proof strategy is analogous to our analysis of the asymptotic average performance ratio [8] but the technical details are quite different because instead of sampling with replacement the harder setting of sampling without replacement has to be analyzed.

Since the random-order ratio of the dual harmonic algorithm DH_k is 0.5 [4], this result separates DNF from DH_k in the random-order model, while their performance cannot be distinguished in a worst-case analysis. This is interesting because DH_k was designed to guard against pathological worst-case inputs and it is already discussed in [4] that one would expect DNF to perform better than DH_k on more realistic inputs. As an additional minor result, we prove that DNF and DH_k are also separated in terms of their asymptotic average performance ratio by showing that this ratio is 0.5 for DH_k while our lower bound of 0.501 for DNF also carries over to this setting.

In contrast to this, we show for bin packing that next-fit, worst-fit, and smart-next-fit do not perform better in the random-order setting and not even in the i.i.d. setting than in the worst case. (For the random-order ratio of next-fit this result was already known [9]).

Class Constrained Bin Packing and Covering. We mention only the most interesting results. For class constrained bin packing we can show that there exists a sequence of algorithms whose asymptotic average performance ratios tend to $h_\infty \approx 1.691$. h_∞ is known in classical bin packing as the lower bound for bounded-space online-algorithms shown by Lee and Lee [12]. This is far better than the competitive ratio of the best known algorithm 2.635 for class constrained bin packing [7] and also beats the known lower bound for arbitrary deterministic algorithms of 1.717 shown in [2] for the special case $k = 2$.

When we consider the random-order model, we find out that several algorithms behave worse than in the case of i.i.d. sampling. Especially, we establish a lower bound of $^{10}/_9$ for the random-order ratio of all deterministic online-algorithms. As far as we know, this is the first lower bound for arbitrary algorithms w.r.t. to a probabilistic performance measure in the area of bin packing and bin covering and its variants.

Furthermore, we consider the special case of unit sized items. We observe again different behaviors of heuristics w.r.t. to the two considered performance measures. Especially, a large class of "natural" algorithms performs asymptotically optimal, if the items are drawn i.i.d.

For class constrained bin covering we investigate the behavior of DNF and FF2. We observe that the algorithms benefit a lot from random input – independently of the considered probabilistic performance measure. We provide bounds, which are logarithmic in k, for the performance of DNF w.r.t. both models. In the case of unit sized items we show that FF2 behaves asymptotically optimal in the random-order model, and therefore also for i.i.d. sampling. We use this result to establish a $^1/_3$-competitive algorithm in the random-order model for general item sizes.

The main tools for proving these results are

- Markov chain arguments (e.g. estimates for the stationary distribution and growth bounds for trajectories);
- couplings to compare stochastic processes and relating i.i.d. sampling with the random-order model;
- concentration inequalities for – possibly dependent – random variables.

An overview on the used concentration bounds will be given in the full version of the paper.

Intuitively one main reason why the two probabilistic measures lead to different results in the case of class constrained bin packing is that in the random-order model the number of different colors can grow with the length of the input sequence while it cannot grow arbitrarily with the input length if the items are drawn i.i.d. with respect to some fixed distribution.

1.4 Algorithms

Let us describe the algorithms that we analyze in more detail. We start with the (class constrained) bin packing problem. For this problem, the following algorithms are relevant for our results.

- Next-Fit (NF): At each point of time one bin is open. NF assigns each arriving item to the currently open bin if it can accommodate the item. Otherwise it

closes the currently open bin and opens a new bin to which the item is added. Here closing a bin means that no item will be assigned to this bin in the future anymore.
- First-Fit (FF): FF never closes a bin, i.e., it keeps all bins open and assigns each arriving item to the first bin that can accommodate it if such a bin exists. Otherwise it opens a new bin and adds the item to it.
- Best-Fit (BF): BF never closes a bin, i.e., it keeps all bins open and assigns each arriving item to the fullest bin that can accommodate it if such a bin exists. Otherwise it opens a new bin and adds the item to it.
- Worst-Fit (WF): WF never closes a bin, i.e., it keeps all bins open and assigns each arriving item to the bin with the most space remaining if this bin can accommodate it. Otherwise it opens a new bin and adds the item to it.
- Smart-Next-Fit (SNF): SNF works similarly to NF. It assigns each arriving item to the currently open bin Z if this bin can accommodate the item. Otherwise it opens a new bin Z' and adds the item to it. It retains as new current bin whichever of Z and Z' has the most space remaining.
- HARMONIC$_M$: HARMONIC$_M$ is an algorithm designed for classical bin packing. It partitions the interval $(0, 1]$ into the subintervals

$$(0, 1/M], (1/M, 1/(M-1)], \dots, (1/2, 1].$$

  This partition induces also a partition of the set of items into M classes. HARMONIC$_M$ packs items from different classes into different bins and it runs NF independently for each class. That is, it packs exactly j items from the interval $(1/(j+1), 1/j]$ into a bin.
- CS [A]: A technique often used to generate algorithms for class constrained bin packing is the ColorSets-approach. The ColorSets-approach wants to apply an algorithm A, which is designed for classical bin packing, to class constrained bin packing. In order to do this, it groups the colors according to their first arrival in groups of size k and then applies A separately to each group. Popular examples are CS [NF], CS [FF] and CS [BF] (see e.g. [14,16]).

While in NF there is only one and in HARMONIC$_M$ only M open bins at each point of time, in FF, BF, and WF all bins are kept open during the whole input sequence. We say that an algorithm is an *ℓ-bounded space algorithm* if on any input and at each point of time it has at most ℓ open bins.

Now we describe the relevant algorithms for the (class constrained) bin covering problem.

- Dual Next-Fit (DNF): DNF packs all arriving items into the same bin until the bin is filled. Then the next items are packed into a new bin until this bin is filled, and so on.
- FF2: The algorithm FF2 is for the class constrained bin covering problem with unit sized items only. It adds each arriving item to the first bin for which it is suitable. To define the notion of suitable, consider a bin that contains already items with $k - t$ different colors. If this bin contains fewer than $B - t$ items, every item is suitable. Otherwise, if the number of items is exactly $B - t$, an item is only suitable if it has a color that is not yet contained in the bin.

– Dual Harmonic DH_M: The algorithm DH_M is the adaption of $\mathrm{HARMONIC}_M$ to classical bin covering. The interval $(0, 1]$ is partitioned into the subintervals $(0, 1/M), [1/M, 1/(M-1)), \dots, [1/2, 1)$. This partition again induces also a partition of the set of items into M classes. DH_M packs items from different classes into different bins and it runs DNF independently for each class. That is, it uses exactly j items from the interval $[1/j, 1/(j-1))$ to cover a bin.

In Sect. 2 we discuss our results on classical bin packing and bin covering in detail, followed by Sects. 3 and 4 on class constrained bin packing and bin covering, respectively.

2 Classical Bin Packing and Covering

We start by showing that the trivial greedy algorithm for the classical bin covering problem performs strictly better in the random-order model than in the worst case. This statement confirms the conjecture given in [4]. Furthermore, to the best of our knowledge this is the first positive result on the random-order ratio of bin packing and covering (and its variations) since the celebrated result of Kenyon [10].

Theorem 1. *We have* $\mathrm{RR}(\mathrm{DNF}) \geq 1/2 + 1/1000$.

DNF is a monotone algorithm in the sense that decreasing the size of items or deleting them only does harm to the algorithm. We see this as follows: Let $I = (a_1, \dots, a_n)$ and $I' = (a_1, \dots, a_{i-1}, a_i', a_{i+1}, \dots, a_n)$ with $a_i' < a_i$. We simulate deleting an item by setting a_i' equal to zero. Let $f(a_j)$ denote the number of the bin a_j is assigned to if DNF performs on I and $f'(a_j)$ if DNF performs on I', respectively. If $j \leq i$ it is obvious that we have $f(a_j) = f'(a_j)$. If $j > i$ we can show via induction that $f(a_j) \geq f'(a_j)$. Therefore, we have $\mathrm{DNF}(I') \leq \mathrm{DNF}(I)$.

It follows from the monotonicity of DNF that we can assume without loss of generality that we deal with instances I that can be packed perfectly into $\mathrm{OPT}(I)$ bins. Especially, we have $\mathrm{OPT}(I) = S(I)$, where $S(I)$ denotes the total size of all items in I.

Since DNF is $1/2$-competitive, we know that the algorithm covers, independently of I^σ, at least $\lfloor \mathrm{OPT}(I)/2 \rfloor$ many bins. For $i \in [1 : \lfloor \mathrm{OPT}(I)/2 \rfloor]$ let $S_i(I^\sigma)$ denote the total size of items in the i-th covered bin if we apply DNF to I^σ. We define the *overshoot* for the i-th bin as $R_i(I^\sigma) := S_i(I^\sigma) - 1$.

Then, the proof of the theorem is based on two pillars. At first, we show that the overshoot is identically distributed:

Lemma 2. *The random variables* $R_i(I^\sigma)$*, where* $1 \leq i \leq \lfloor \mathrm{OPT}(I)/2 \rfloor$*, are identically distributed.*

Then, we can express the random-order ratio in terms of the overshoot. Let $W(I^\sigma)$ denote the total size of the items in the last bin, which is not covered. We have

$$\begin{aligned}
\mathrm{OPT}(I) &= \mathrm{DNF}(I^\sigma)\cdot 1 + \sum_{i=1}^{\mathrm{DNF}(I^\sigma)} R_i(I^\sigma) + W(I^\sigma) \\
&= \mathrm{DNF}(I^\sigma) + \sum_{i=1}^{\lfloor \mathrm{OPT}(I)/2 \rfloor} R_i(I^\sigma) + \sum_{i=\lfloor \mathrm{OPT}(I)/2\rfloor+1}^{\mathrm{DNF}(I^\sigma)} R_i(I^\sigma) + W(I^\sigma) \\
&\le \mathrm{DNF}(I^\sigma) + \sum_{i=1}^{\lfloor \mathrm{OPT}(I)/2 \rfloor} R_i(I^\sigma) + (\mathrm{DNF}(I^\sigma) - \lfloor \mathrm{OPT}(I)/2 \rfloor) + 1 \\
&\le 2\,\mathrm{DNF}(I^\sigma) - \mathrm{OPT}(I)/2 + 2 + \sum_{i=1}^{\lfloor \mathrm{OPT}(I)/2 \rfloor} R_i(I^\sigma).
\end{aligned}$$

Applying expectation values to both sides and using the previous lemma, we obtain

$$\mathrm{OPT}(I) \le 2\mathbb{E}\left[\mathrm{DNF}(I^\sigma)\right] - \mathrm{OPT}(I)/2 + 2 + \frac{1}{2}\,\mathrm{OPT}(I)\mathbb{E}\left[R_1(I^\sigma)\right].$$

It follows that

$$\frac{\mathbb{E}\left[\mathrm{DNF}(I^\sigma)\right]}{\mathrm{OPT}(I)} \ge \frac{3}{4} - \frac{1}{4}\cdot\mathbb{E}\left[R_1(I^\sigma)\right] - \frac{1}{\mathrm{OPT}(I)}. \tag{1}$$

The second pillar is to give an upper bound for the overshoot. A similar statement in case of items that are drawn i.i.d. was shown in our paper [8]. At that time we used elementary counting and covering arguments. This time we apply concentration inequalities that lead to a simplified proof with a stronger bound – even if the resulting bound is still close to the worst case.

Lemma 3. *Let $(I_j)_j$ be an arbitrary sequence of instances with*

$$\lim_{j\to\infty} \mathrm{OPT}(I_j) = \infty \quad \textit{and} \quad \mathrm{RR}(\mathrm{DNF}) = \liminf_{j\to\infty} \frac{\mathbb{E}\left[\mathrm{DNF}(I_j^\sigma)\right]}{OPT(I_j)}.$$

Then, if j is sufficiently large we have

$$\mathbb{E}\left[R_1(I_j^\sigma)\right] \le 1 - \frac{34}{100e^3}\cdot\left(1-\exp\left(-\frac{121}{420}\right)\right) \approx 0.99576.$$

Combining this upper bound with (1) yields a lower bound of approximately 0.501 for DNF if the items arrive in random order. The lower bound is complemented by an upper bound of 2/3 for the random-order ratio in [8].

The behavior of DNF in the random-order model is in contrast to the behavior of NF in classical bin packing: Coffman et al. [9] showed in 2008 that RR(NF) = 2, which is equal to its worst-case performance. We will refine this statement and show that the algorithms NF, SNF and WF for classical bin packing do not behave better than in the worst case even if the items are sampled in an i.i.d. manner.

Proposition 4. *For* $\mathrm{A} \in \{\mathrm{NF}, \mathrm{SNF}, \mathrm{WF}\}$ *we have* $\mathrm{AAPR}(\mathrm{A}) = \mathrm{RR}(A) = 2$.

Furthermore, also the dual harmonic algorithm DH_k for bin covering does not improve on the worst case if the items are drawn i.i.d.

Proposition 5. *We have* $\mathrm{AAPR}(\mathrm{DH}_k) = 1/2$.

3 Class Constrained Bin Packing

3.1 Results for General Item Sizes

As already mentioned in the introduction a popular approach to deal with class constrained bin packing is the ColorSets-approach. We show that there are algorithms based on this approach, that behave remarkably well in the case of items that are drawn i.i.d. Let $t_1 = 1$ and $t_{i+1} = t_i(t_i + 1)$. We set $\sum_{i=1}^{\infty} \frac{1}{t_i} =: h_\infty \approx 1.691$. h_∞ is known in bin packing as the famous lower bound for bounded-space online algorithms for the classical bin packing problem proved by Lee and Lee in [12]. The following statement shows that there is a sequence of ColorSets-based algorithms whose performance tends to h_∞. Furthermore, no algorithm based on this idea could behave better.

Theorem 6. *Let* $\epsilon > 0$ *be arbitrary. Choosing* M *sufficiently large, we have*

$$\mathrm{AAPR}(\mathrm{CS}\,[\mathrm{HARMONIC}_M]) \leq h_\infty + \epsilon.$$

Furthermore, let A *be an arbitrary algorithm for classical bin packing, then we have*

$$\mathrm{AAPR}(\mathrm{CS}\,[A]) \geq h_\infty.$$

To show the upper bound we want to compare the performance of the algorithm $\mathrm{CS}\,[\mathrm{HARMONIC}_M]$ with the performance of the algorithm $\mathrm{HARMONIC}_M$ in the case we ignore the colors. Lee and Lee proved in [12] that the asymptotic competitive ratio of $\mathrm{HARMONIC}_M$ for classical bin packing is upper bounded by $h_\infty + \epsilon$ for $\epsilon > 0$ arbitrary, if we choose M sufficiently large.

We observe that the number of opened bins differs by at most $M \cdot Q_F(n)$ many bins. Here, $Q_F(n)$ denotes the number of different drawn colors among the first n drawn items. But $Q_F(n)$ grows sublinearly in expectation. Therefore, asymptotically the performance of $\mathrm{CS}\,[\mathrm{HARMONIC}_M]$ coincides with the performance of $\mathrm{HARMONIC}_M$ in classical bin packing.

For the lower bound we construct a distribution F as follows: The multiset of items $\mathcal{I}$ contains *large* items and *small* items. The large items are as follows: For each $(i, j) \in [k]^2$ there will be an item of size $\frac{1}{t_i+1} + \beta$ of color $(i-1)k + j$, where $\beta > 0$ is sufficiently small. Furthermore, $\mathcal{I}$ contains lots of small items of different colors. Choosing the small items appropriately the order of the first arrival of the colors is $1, \ldots, k^2$ with high probability. Then, $\mathrm{CS}\,[A]$ will pack the items of size $\frac{1}{t_i+1} + \beta$ separately. The statement then follows from the work of Lee and Lee.

In the random-order model things are more complicated: We can show that it is not possible for ColorSets-based algorithms and FF to achieve a performance of h_∞.

Proposition 7. *Let A be an arbitrary algorithm for classical bin packing. Then we have* $\mathrm{RR}(\mathrm{CS}\,[A]) \geq 2$*, even in the special case* $k = 2$*. Furthermore, we show that* $\mathrm{RR}(\mathrm{FF}) \geq 2$*.*

Furthermore, we are able to establish a non-trivial lower bound for the performance of an arbitrary online-algorithm in the random-order model. As far as we know this is the first asymptotic lower bound for a probabilistic performance measure in the field of bin packing/bin covering and its variants.

Theorem 8. *Let A be an arbitrary online-algorithm for class constrained bin packing. Then we have* $\mathrm{RR}(A) \geq {}^{10}/_{9}$*.*

The idea of the proof is to construct an instance I that contains items of colors of the two types *small* and *large*. The total size of all items of a small color is close to zero, while the total size of all items of a large color is close to 1. Furthermore, for each color there are lots of tiny items. If the items arrive in random order, there will be lots of tiny items in the beginning. This forces the algorithm to decide which colors to put in the same bin without the possibility to learn, which color is small and large. Therefore, there will be a constant fraction of bins opened by the algorithm that are nearly empty.

3.2 The Special Case of Unit Sized Items

Now we want to consider the case of unit sized items. We observe the same behavior of algorithms as in the case of general item sizes. If the items are drawn i.i.d. a large class of natural algorithms performs asymptotically optimal, but in the random-order model their performance is worse.

Proposition 9. $\mathrm{CS}\,[\mathrm{NF}]$ *and every algorithm that opens a new bin only if it is forced, is optimal if the items are drawn i.i.d.*

Proposition 10. *We have* $\mathrm{RR}(\mathrm{CS}\,[\mathrm{NF}]) = 2$*.*

Proposition 11. *We have* $\mathrm{RR}(\mathrm{FF}) \geq 1.5$*.*

Finally, we want to mention that bounded-space algorithms perform poorly for class constrained bin packing, even on random input with unit sized items. This is in contrast to classical bin packing.

Proposition 12. *Consider class constrained bin packing with unit sized items and parameters B and k. Let A be an arbitrary bounded-space online-algorithm. Then we have* $\mathrm{RR}(A), \mathrm{AAPR}(A) \in \Omega(B/k)$*.*

4 Class Constrained Bin Covering

We start with a simple result on bounded-space algorithms for the class constrained bin covering problem. While in the classical bin covering problem, even the trivial 1-bounded space algorithm is best possible w.r.t. the competitive ratio, in the class constrained variant those algorithms behave poorly.

Proposition 13. *Let A be a bounded-space algorithm. Then A is not competitive w.r.t. the competitive ratio.*

In general, there is a logarithmic upper bound in k for the performance of online algorithms. That is, the online version of this problem is strictly more difficult than the classical problem. The following statement is a slight improvement on the corresponding result in [6]. The proof uses the same technique, but adjusts the choice of scenarios.

Proposition 14. *The competitive ratio of any deterministic online algorithm is at most $\left(H_{k-1} + 1 - \frac{k-1}{B}\right)^{-1}$. If $B = k$ this yields an upper bound of H_k^{-1}.*

Now we begin to investigate the performance of heuristics w.r.t. probabilistic performance measures. We start with the simple 1-bounded space algorithm DNF.

Theorem 15. *For unit sized items we have* $\mathrm{RR}(\mathrm{DNF}) \in \Theta(\log(k)^{-1})$*. For general item sizes we have* $\mathrm{AAPR}(\mathrm{DNF}) \in \Theta(\log(k)^{-1})$*.*

We see that in class constrained bin covering DNF benefits a lot from probabilistic input. We have seen that bounded-space algorithms cannot be competitive in the worst case and that there exists a logarithmic upper bound in k for the performance of arbitrary online algorithms. If unit sized items arrive in random order, even the simple 1-bounded-space algorithm DNF achieves a competitive ratio that matches this bound.

Surprisingly it turns out that a FirstFit-approach is even optimal if unit sized items arrive in random order:

Theorem 16. *We have* $\mathrm{RR}(\mathrm{FF2}) = 1$*.*

To prove the theorem at first we observe that the algorithm is monotone. Therefore, we can assume that we deal with instances I that cover $\mathrm{OPT}(I)$ bins perfectly. Furthermore, the monotonicity of FF2 allows us to restart the algorithm several times starting again with a single empty bin. Using this technique, we divide the input I into $|I|^{2/3}$ many sub-inputs containing each $|I|^{1/3}$ many items. Then we show that we can assume that the items in the sub-input are drawn i.i.d. So we reduce the analysis in the random-order model to the case of i.i.d. sampling.

Then, we construct a comparison Markov chain, which lower bounds the number of covered bins of FF2 on the sub-inputs. The idea of the comparison chain is as follows: We simulate the behavior of a modified FF2-algorithm on a special distribution F. We obtain F from drawing an item with color 1 with probability $(B - k + 1)/B$ and an item with color i with probability $1/B$, where $i \in \{2, \ldots, k\}$. The modified FF2-algorithm treats the first $B - k + 1$ items in a bin as items of color 1. We can show that FF2 on an arbitrary (perfect-packing) distribution F covers in expectation at least as many bins as the modified algorithm in our comparison chain.

Finally, we use tools from the field of Markov chains to show that the growth of open bins in the comparison chain is only sublinear in the number of items. Plugging the pieces together we obtain that RR(FF2) $= 1$.

Theorem 16 allows us to give a simple online algorithm, which is $1/3$-competitive for general item sizes if the items arrive in random-order.

Corollary 17. *There exists an $1/3$-competitive algorithm in the random-order model for the class constrained bin covering problem with general item sizes.*

This also gives us an easy randomized algorithm for the offline case. To the best of our knowledge this is the first offline algorithm presented for this problem.

Corollary 18. *There is a randomized asymptotic $1/3$-competitive algorithm for class constrained bin covering in the offline case.*

5 Conclusion and Further Research

We showed that the DNF algorithm for bin covering performs better in the random-order model than in the worst case by providing a lower bound of 0.501 on its random-order ratio. This is the first bound better than the trivial bound of 0.5. We think that it is an interesting open problem to close the gap between the lower and upper bounds and we conjecture that the random-order ratio of DNF equals the upper bound 2/3.

Furthermore, we studied class constrained bin packing and class constrained bin covering in the random-order model and i.i.d. sampling. We saw that in many cases heuristics benefit from the probabilistic input and can beat several worst-case bounds. In class constrained bin packing we observed different performances of algorithms in the two probabilistic models. The random-order model allows us to restrict the number of similar items and to force a linear number of different item types, while in the i.i.d. model the number of different item types grows only sublinearly. This difference plays an important role in class constrained bin packing, while it is not relevant in class constrained bin covering. As far as we know in bin packing and bin covering there are no other results known, where the performance of algorithms differ with respect to the investigated performance measures. It would be interesting to find further examples in this area, where algorithms perform differently, and to give formal explanations why both performance measures coincide in other bin packing/bin covering variants.

References

1. Babel, L., Chen, B., Kellerer, H., Kotov, V.: Algorithms for on-line bin-packing problems with cardinality constraints. Discrete Appl. Math. **143**, 238–251 (2004)
2. Balogh, J., Békési, J., Dósa, G., Epstein, L., Levin, A.: Lower bounds for several online variants of bin packing. CoRR, abs/1708.03228 (2017)
3. Balogh, J., Békési, J., Dósa, G., Epstein, L., Levin, A.: Online bin packing with cardinality constraints resolved. In: Proceedings of the 25th European Symposium on Algorithms (ESA), vol. 87, pp. 10:1–10:14 (2017)
4. Christ, M.G., Favrholdt, L.M., Larsen, K.S.: Online bin covering: expectations vs. guarantees. Theoret. Comput. Sci. **556**, 71–84 (2014)
5. Dósa, G., Sgall, J.: Optimal analysis of best fit bin packing. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 429–441. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_36
6. Epstein, L., Imreh, C., Levin, A.: Class constrained bin covering. Theory Comput. Syst. **46**(2), 246–260 (2010)
7. Epstein, L., Imreh, C., Levin, A.: Class constrained bin packing revisited. Theoret. Comput. Sci. **411**(34–36), 3073–3089 (2010)
8. Fischer, C., Röglin, H.: Probabilistic analysis of the dual next-fit algorithm for bin covering. In: Kranakis, E., Navarro, G., Chávez, E. (eds.) LATIN 2016. LNCS, vol. 9644, pp. 469–482. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49529-2_35
9. Coffman Jr., E.G., Csirik, J., Rónyai, L., Zsbán, A.: Random-order bin packing. Discrete Appl. Math. **156**(14), 2810–2816 (2008)
10. Kenyon, C.: Best-fit bin-packing with random order. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 359–364 (1996)
11. Krause, K.L., Shen, V., Schwetman, H.D.: Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J. ACM **22**(4), 522–550 (1975)
12. Lee, C.C., Lee, D.T.: A simple on-line bin-packing algorithm. J. ACM **32**(3), 562–572 (1985)
13. Shachnai, H., Tamir, T.: Polynomial time approximation schemes for class-constrained packing problems. J. Sched. **4**(6), 313–338 (2001)
14. Shachnai, H., Tamir, T.: Tight bounds for online class-constrained packing. Theoret. Comput. Sci. **321**(1), 103–123 (2004)
15. Ullman, J.D.: The Performance of a Memory Allocation Algorithm. Technical report 100 (Princeton University, Department of Electrical Engineering, Computer Sciences Laboratory). Princeton University (1971)
16. Xavier, E.C., Miyazawa, F.K.: The class constrained bin packing problem with applications to video-on-demand. Theoret. Comput. Sci. **393**(1–3), 240–259 (2008)

Locating the Eigenvalues for Graphs of Small Clique-Width

Martin Fürer[1](✉), Carlos Hoppen[2], David P. Jacobs[3], and Vilmar Trevisan[2]

[1] Department of Computer Science and Engineering, Pennsylvania State University, State College, USA
furer@cse.psu.edu

[2] Instituto de Matemática, Universidade Federal do Rio Grande do Sul, Alegre, Brazil
choppen@ufrgs.br, trevisan@mat.ufrgs.br

[3] School of Computing, Clemson University, Clemson, USA
dpj@clemson.edu

Abstract. It is shown that if G has clique-width k, and a corresponding tree decomposition is known, then a diagonal matrix congruent to $A - cI$ for constants c, where A is the adjacency matrix of the graph G of order n, can be computed in time $O(k^2 n)$. This allows to quickly tell the number of eigenvalues in a given interval.

Keywords: Eigenvalues · Clique-width · Congruent matrices
Efficient algorithms · Parameterized algorithms

1 Introduction

Throughout this paper we use standard terminology for graph theory and linear algebra. The main concern of spectral graph theory is to determine properties of a graph through the eigenvalues of matrices associated with it. An obvious step in any such application is to calculate the spectrum of the input graph, or at least to accurately estimate a subset of its eigenvalues. We say that an algorithm *locates eigenvalues* if, for any graph G and any real interval I, it finds the number of eigenvalues of G in the interval I. In recent years, efficient algorithms have been developed for the location of eigenvalues in trees [8], unicyclic graphs [3], threshold graphs [9] (also called *nested split graphs*), and chain graphs [1]. A slightly richer class of graphs which contain threshold graphs are the graphs with no induced subgraph isomorphic to P_4, which are often called P_4*-free* graphs or *cographs*. Eigenvalue location in cographs and threshold graphs has been widely studied [2,10,16,19–21].

A powerful concept for parameterized algorithms is *clique-width* [4–6]. Its motivation has been to extend the concept of tree-width introduced by Robertson

M. Fürer—Research supported in part by NSF Grant CCF-1320814.
D. P. Jacobs and V. Trevisan—Work supported by Science without Borders CNPq - Grant 400122/2014-6, Brazil.

and Seymour [18] (e.g., see [17]) to include dense graphs as well. A *k-expression* is an expression formed from atoms $i(v)$, two types of unary operations $\eta_{i,j}$ and $\rho_{i \to j}$, and a binary operation $\oplus$ as follows.

- $i(v)$ creates a vertex v with label i, where i is from the set $[k] = \{1, \ldots, k\}$.
- $\eta_{i,j}$ creates all edges which are not already present between the vertices with labels i and the vertices with labels j for $i \neq j$.
- $\rho_{i \to j}$ changes all labels i to j.
- $\oplus$ produces the disjoint union of two labeled graphs.

Finally, the graph generated by a k-expression is obtained by deleting the labels. The *clique-width $cw(G)$* of a graph G is the smallest k such that the graph can be defined by a k-expression [5,6].

Any graph can be constructed in this way, provided k is large enough. For instance, cographs are exactly the graphs for which $cw(G) \leq 2$, and one can show that $cw(T) \leq 3$ for any tree T. See [14] for a discussion of the clique-width of many classical classes of graphs. Computing the clique-width is NP-hard [7]. Thus, one usually assumes that a graph is given together with a k-expression. For most practical applications, including ours, a constant factor approximation would be sufficient to obtain efficient algorithms. The complexity of finding such an approximation is still unknown. This will hopefully improve in the future, but one should note that even the tree-width is NP-hard to compute, and no polynomial time approximation algorithm is known.

While tree-width has always been related to symmetric Gaussian elimination (a polynomial time algorithm), the main application area for graph widths has been the design of efficient algorithms for NP-complete or even harder problems. The goal there is to show problems fixed parameter tractable (FPT), by providing an algorithm with a running time of $O(f(k)n^c)$ for an arbitrary computable function f. Typically, f is at least exponential. But for small values of k, such algorithms are often very practical.

Here, we return to a polynomial time solvable problem. Nevertheless, the parameterized complexity view is very useful. For bounded clique-width, we turn a cubic time solution (or at least more than quadratic time solution when using known fast matrix multiplication) into a linear time solution, a drastic improvement for graphs of small clique-width, even though the transformation is not at all straightforward.

Recall that two matrices R and S are *congruent*, which we write $R \cong S$, if there exists a nonsingular matrix P for which $R = P^T SP$. It turns out that there is a strong connection between congruence of symmetric matrices and eigenvalue location, which we now describe. Let G be a graph with adjacency matrix A, and consider real numbers c and d with $c < d$. If we can construct a diagonal matrix $D_c \cong B = A - cI$, then Sylvester's Law of Inertia [15, p. 568] implies that the number n_1 of eigenvalues of A greater than c (counted with their multiplicities) equals the number of positive entries in D_c. Similarly, the number n_2 of eigenvalues of A greater than d is the number of positive entries in a diagonal matrix $D_d \cong A - dI$. Thus $n_1 - n_2$ is the number of eigenvalues (counted with their multiplicities) in $(c, d]$. This is why we want to design a fast algorithm to find a diagonal matrix that is congruent to $A - cI$.

The purpose of this paper is to give an $O(k^2 n)$ *time diagonalization algorithm for graphs having* clique-width k. This is remarkable because adjacency matrices of graphs with clique-width k often have $\Omega(n^2)$ nonzero entries. In particular, graphs of bounded clique-width are often not of bounded tree-width. The new clique-width based algorithm can be much more efficient than a tree-width based algorithm, because the tree-width can be linear in n, even when the clique-width is a small constant.

Closely related to clique-width is the lesser known *NLC-width*, due to Wanke [22], and initiated by node label controlled (NLC) graph grammars [11,12]. Graphs of NLC-width at most k are defined by NLC k-expressions. These expressions contain the operators $i(v)$ and $\rho_{i \to j}$ for vertex creation and relabeling. But new edges are created in *combination* with the join operation, using a binary operation $\oplus_S$ (instead of $\oplus$ and $\eta_{i,j}$), where $S \subseteq [k] \times [k]$. When $G \oplus_S H$ is applied, then, for each $(i, j) \in S$, edges are introduced between vertices labeled i in G and vertices labeled j in H. This has the effect that a subgraph generated by a subexpression is always an induced subgraph, a property important to us. This is different from k-expressions defining clique-width, where an edge creating operation applied after a join of G and H typically introduces edges within G and within H too.

In representing graphs, we will actually use a minor modification of NLC-width, which is much more convenient than clique-width for our purposes. We call it *slick clique-width*. Here a single operator performs the join, edge creation and relabeling. It also has the property that subgraphs generated by subexpressions are induced subgraphs.

The remainder of this paper is organized as follows. In Sect. 2 we define slick k-expressions and slick clique-width, denoted $scw(G)$. We also show that $scw(G) \leq cw(G) \leq 2scw(G)$ and observe that there are linear-time transformations to translate a k-expression to a slick k-expression, and a slick k-expression to a $2k$-expression. In Sect. 3 we describe our $O(k^2 n)$ time diagonalization algorithm for graphs of slick clique-width k. Concluding remarks appear in Sect. 4.

2 Slick Clique-Width

In the following definition a single operator is used for performing the union, creating edges and relabeling. A *slick* k*-expression* is an expression formed from atoms $i(v)$ and a binary operation $\oplus_{S,L,R}$, where L, R are functions from $[k]$ to $[k]$ and S is a binary relation on $[k] \times [k]$, as follows.

(a) $i(v)$ creates a vertex v with label i, where $i \in [k]$.
(b) Given two graphs G and H whose vertices have labels in $[k]$, the labeled graph $G \oplus_{S,L,R} H$ is obtained by the following operations. Starting with the disjoint union of G and H, add edges from every vertex labeled i in G to every vertex labeled j in H for all $(i, j) \in S$. Afterwards, every label i

of the *left component* G is replaced by $L(i)$, and every label i of the *right component* H is replaced by $R(i)$.

Two (slick) clique-width expressions are said to be *equivalent* if they produce the same labeled graph. Finally, the *graph generated by a slick clique-width expression* is obtained by deleting the labels of the labeled graph produced by it. The *slick clique-width* $scw(G)$ of a graph G is the smallest k such that the graph can be defined by a slick k-expression.

Note that the definition of a slick k-expression implies that edges can only be placed between different components, so if two vertices are in the same component after some steps in the above construction, but are not adjacent, they will never become adjacent. A slick k-expression can also be represented as a parse tree T where the leaves contain the operators $i(v)$ and the internal nodes contain the $\oplus_{S,L,R}$ operations. Two vertices v and w are adjacent if and only if their least common ancestor $\oplus_{S,L,R}$ in T connects them, similar to the cotree representation for cographs in [2]. Note that cographs are precisely the graphs with slick clique-width equal to 1. Indeed, recall that G is a cograph if and only if either G is a single vertex or the union $G_1 \cup G_2$ or join $G_1 \otimes G_2$ of cographs G_1 and G_2 (see [2]). On the other hand, when there is a single label available, the functions L and R are trivial identities, so that $\oplus_{S,L,R}$ either creates a disjoint union (if $S = \emptyset$) or adds all possible edges with ends in the two operands (if $S = \{(1,1)\}$). Another simple argument shows that $scw(T) \leq 2$ for any tree T.

We can define the *depth* $d(r)$ of a slick k-expression r recursively. The expression $i(v)$ has depth 0, and $d(A \oplus_{S,L,R} B) = 1 + \max\{d(A), d(B)\}$. This is equivalent to the depth of the parse tree for r. In a similar way, we can define the depth of a k-expression.

The next result shows that the concept of clique-width and slick clique-width are closely related. See [13] for the corresponding inequalities for clique-width and NLC-width.

Theorem 1. *If G is a graph then $scw(G) \leq cw(G) \leq 2scw(G)$.*

Proof. This follows from Lemmas 1 and 3 below. □

Lemma 1. *If G is a graph then $scw(G) \leq cw(G)$.*

Proof. As an auxiliary tool to transform k-expressions into slick k-expressions, we introduce a new unary operator $o_{S,L}$ that works similar to an $\oplus_{S,L,R}$, except that there is no join operation. We assume, S is a symmetric irreflexive binary relation, $L : [k] \to [k]$ is a function, and s is a k-expression generating a graph G. Then, for all i, j with $(i,j) \in S$, $o_{S,L}(s)$ adds edges from all vertices labeled i to all vertices labeled j in G. Thereafter every label i is replaced by $L(i)$.

We show that for every k-expression s the expression $o_{S,L}(s)$ is equivalent to a slick k-expression. The proof is by induction on the depth of s. The basis is trivial, because $o_{S,L}(i(v))$ is equivalent to $L(i)v$. For the induction step, we do a case distinction according to the outermost operation in s.

– Let $s = \eta_{i,j}(s')$. The expression $o_{S,L}(\eta_{i,j}(s'))$ transforms into the equivalent expression $o_{S',L'}(s')$ with $S' = S \cup \{(i,j)\}$ and $L' = L$.

– Let $s = \rho_{i \to j}(s')$. For
$$\rho(i') = \begin{cases} j \text{ if } i' = i \\ i' \text{ otherwise,} \end{cases}$$
the expression $o_{S,L}(\rho_{i \to j}(s'))$ transforms into the equivalent expression $o_{S',L'}(s')$ with
$$(i', j') \in S' \text{ iff } (\rho(i'), \rho(j')) \in S,$$
and
$$L'(i') = L(\rho(i')) = \begin{cases} L(j) \text{ if } i' = i \\ L(i') \text{ otherwise.} \end{cases}$$
Note that if the relations S and S' are represented by symmetric 0–1-matrices, then S' is obtained from S by copying the j-th row into its i-th row, and the j-th column into the i-th column. The order of the two copying operations does not matter. In any case, we obtain $S'_{jj} = S'_{ij} = S'_{ji} = S'_{ii} = S_{jj}$.
– Let $s = s' \oplus s''$. The expression $o_{S,L}(s' \oplus s'')$ transforms into the equivalent expression $(o_{S',L'}s') \oplus_{S,L,L} (o_{S',L'}s'')$, where $S' = S$ and L' is the identity function.

In all cases, the k-expressions s', and also s'' in the last case, are of smaller depth than s. Hence, by the induction hypothesis, $o_{S',L'}s'$ and $o_{S',L'}s''$ are equivalent to slick k-expressions.

We obtain the complete transformation of a k-expression s into a slick k-expression s' by transforming the expression $o_{S,L}(s)$ for a dummy operation $o_{S,L}$ with S the empty relation and L the identity function. □

Lemma 2. *A transformation of a k-expression s into a slick k-expression s' can be obtained in time $O(k^2 n + |s|)$.*

Proof. The procedure described in the proof of Lemma 1 runs in time $O(k^2|s|)$. It spends time $O(k^2)$ in every node of the parse tree. This is not good, if we have a large number of nodes.

First, we observe that the number of leaves in the parse tree of s is n, and the number of branching nodes is $n-1$. Between them there could be arbitrary long sequences of $\eta_{i,j}$ and $\rho_{i \to j}$ operations, even though only $O(k^2)$ of them would be meaningful in each sequence. We would obtain the desired running time of $O(k^2 n + |s|)$ if we only spent time $O(1)$ in the $\eta_{i,j}$ and $\rho_{i \to j}$ nodes. This is easily achieved for the $\eta_{i,j}$ operations, as they only require to change one entry in a $k \times k$ matrix representing S. A $\rho_{i \to j}$ operation asks for an $O(1)$ change in L, and also for a row and column operation in the matrix representing S. We can afford $O(k)$ such operations in each sequence t of $\eta_{i,j}$ and $\rho_{i \to j}$ operations between branching and leaf operations.

If the length of the given k-expression s is more than $O(kn)$, then we need a better algorithm to transform a k-expression s into a slick k-expression s'. In a first phase, we preprocess every sequence t between nodes p and q where p is leaf or a branching node and q is the next branching node. While walking up from p to q towards the root, we determine a set of unused labels. In p, we might assume that all labels are used. After seeing a $\rho_{i \to j}$, we record label i as unused. We put all the $\rho_{i \to j}$ operations into 3 classes.

- If label i is already unused when we reach a $\rho_{i \to j}$ operation, then this operation is useless and is immediately discarded. It would have no effect, because there are no vertices labeled i.
- If label i is used, but j is unused, then we classify a $\rho_{i \to j}$ operation as a relabeling.
- If both labels i and j are used, then we classify a $\rho_{i \to j}$ operation as a proper merge.

Between p and q there is an arbitrary number of relabeling operations. In order to handle each in time $O(1)$ instead of $O(k)$, we access the matrix S via an index function $\rho : [k] \to [k]$. An $\eta_{i,j}$ operation sets $S_{\rho(i)\rho(j)}$ and $S_{\rho(j)\rho(i)}$ to 1 instead of S_{ij} and S_{ji}. A relabeling operation $\rho_{i \to j}$ now just redefines $\rho(i)$ to $\rho(j)$, instead of copying the $\rho(j)$-th row and column into the $\rho(i)$-th row and column. The effect is the same, because before the $\rho_{i \to j}$ operation, there were no vertices labeled j.

Between p and q there are at most $k-1$ proper merge operations. Each is handled in time $O(k)$ by copying a row and column of S.

Now we show that the new algorithm is correct. We extend $o_{S,L}$ to $o_{\rho,S,L}$.

- For all k-expressions s, $o_{\rho,S,L}\, \eta_{i,j}\, s$ is equivalent to $o_{\rho,S',L}\, s$, where $S' = S \cup \{(\rho(i), \rho(j)), (\rho(j), \rho(i))\}$. In both cases, the only difference to $o_{\rho,S,L}\, s$ is that the edges between vertices labeled i and j are added.
- Equally straightforward is the fact that for all k-expressions s' and s'', $o_{S,L}(s' \oplus s'')$ is equivalent to $(o_{S',L'}\, s') \oplus_{S,L,L} (o_{S',L'}\, s'')$.
- For all k-expressions s, if the first $\rho_{i \to j}$ in $o_{\rho,S,L}\, \rho_{i \to j}\, s$ is a relabeling operation, then $o_{\rho,S,L}\, \rho_{i \to j}\, s$ is equivalent to $o_{\rho',S,L'}\, s$, where $\rho'(i') = \rho(i')$ except for $\rho'(i) = \rho(j)$, and $L'(i') = L(i')$ except for $L'(i) = L(j)$. In both cases, the only difference to $o_{\rho,S,L}\, s$ is that label i is changed to label j. Edges not involving label i are still created the same way. $o_{\rho,S,L}\, \rho_{i \to j}$ creates edges between label i and some label $i' \neq i$ if and only if $(\rho(j), \rho(i')) \in S$. Noting that $\rho(j) = \rho'(i)$ and $\rho(i') = \rho'(i')$ for $i' \neq i$, we see that $o_{\rho',S,L'}\, s$ creates the same edges. As $o_{\rho,S,L}\, \rho_{i \to j}\, s$ and $o_{\rho',S,L'}\, s$ also create the same labels, they are equivalent.
- For all k-expressions s, if the first $\rho_{i \to j}$ in $o_{\rho,S,L}\, \rho_{i \to j}\, s$ is a proper merge, then it is equivalent to $o_{\rho,S',L'}\, s$, where S' is obtained from S by copying the j-th row and column into the i-th row and column, and $L'(i') = L(i')$ except for $L'(i) = L(j)$. In both cases, edges between labels i' and j' are created if $(\rho(i'), \rho(j')) \in S'$. Furthermore, the label replacements are the same in both cases. $\square$

Lemma 3. *If G is a graph then $cw(G) \leq 2scw(G)$.*

Proof. Suppose $k = scw(G)$, and s is a slick k-expression for G. It suffices to construct an equivalent $2k$-expression r. We will show this by induction on the depth of s. This is clear when $s = i(v)$. Otherwise, let $s = s_1 \oplus_{S,L,R} s_2$, for slick k-expressions s_1 and s_2. By the induction hypothesis, we can assume that each slick k-expression s_i has an equivalent $2k$-expression r_i, producing the same labeled graph. So we have

$$s = r_1 \oplus_{S,L,R} r_2.$$

Note that, even though r_1 and r_2 may involve labels between $k+1$ and $2k$, they produce labeled graphs with labels in $[k]$. To complete the induction we translate the behavior of $\oplus_{S,L,R}$ into the operators $\rho_{i \to j}$, $\eta_{i,j}$ and $\oplus$. We relabel the vertices on the left side mapping each label $i \in [k]$ to $i+k$, and then form the union. Suppressing parentheses, we obtain a subexpression

$$w = (\rho_{1 \to k+1} \quad \rho_{2 \to k+2} \quad \dots \quad \rho_{k \to 2k} \quad r_1) \oplus r_2. \tag{1}$$

To obtain edges, for each $(i,j) \in S$ we apply the operators $\eta_{i+k,j}$ to w, obtaining a new expression w'. The relabeling in (1) ensures that new edges are placed *only* between the left and right sides. Depending on the functions L and R, we finally relabel the left and right sides back to their desired values in $[k]$.

We need to relabel according to the function $f : [2k] \to [2k]$ defined by

$$f(i) = \begin{cases} L(i-k) & \text{if } i > k \\ R(i) & \text{if } i \leq k. \end{cases}$$

We do this relabeling in three rounds.

First, we reduce the number of labels to at most k, using the $2k$-expression

$$w' = \rho_{1 \to g(1)} \cdots \rho_{2k \to g(2k)} w$$

where

$$g(i) = \max\{j \mid f(i) = f(j)\}.$$

Note that the set $\{j \mid f(i) = f(j)\}$ is not empty, as it contains the label i.

Now, we move the set of current labels $I = \{i_1, \dots, i_q\}$ with $i_1 < i_2 < \cdots < i_q$ and $q \leq k$ up into the interval $[2k-q+1, 2k]$ using the $2k$-expression

$$w'' = \rho_{i_1 \to h(i_1)} \cdots \rho_{i_q \to h(i_q)} w'$$

where $h : I \to [2k-q+1, 2k]$ is given by $h(i_j) = 2k - q + j$. Note that the break down into the previous 2 rounds is just to simplify the description. These relabelings could have been combined into one round.

In the third round, we choose the proper new labels with the $2k$-expression

$$w''' = \rho_{2k-q+1 \to f(h^{-1}(2k-q+1))} \, \rho_{2k-q+2 \to f(h^{-1}(2k-q+2))} \cdots \rho_{2k \to f(h^{-1}(2k))} \, w''$$

Finally, we delete all the $\rho_{i \to i}$ operations, or declare them as having no effect.

$\square$

The cograph diagonalization algorithm in [10] exploited the fact that in any cograph of order $n \geq 2$, there exist two vertices u and v for which $N(u) = N(v)$ or $N[u] = N[v]$, so-called siblings. ($N(v)$ is the set of neighbors of v, and $N[v] = N(v) \cup \{v\}$ is the closed neighborhood of v.) This means that their corresponding rows and columns in the adjacency matrix can differ by at most two positions. By subtracting say, the row (column) of u from the row (column) of v, the row and column of v is annihilated except in one off diagonal position. The following analog is crucial to our algorithm.

Remark 1. Let T_G be a parse tree for a graph G with adjacency matrix A, and Q a node in T_G. If two vertices u and v have the same label at Q, then their rows (columns) will agree outside of the matrix for the subtree rooted at Q.

Two symmetric matrices are congruent if one can be obtained by the other by a sequence of *pairs* of elementary operations, each pair consisting of a row operation followed by the *same* column operation. In our algorithm we only use congruence operations that add a multiple of a row or column to another row or column respectively. To achieve linear-time we must operate on a sparse representation of the graph, rather than the adjacency matrix.

3 The Algorithm

We are given a graph $G = (V, E)$ of slick clique-width k given by a slick k-expression Q_0. The generated graph G has $n = |V|$ vertices, and the slick k-expression Q_0 has length $O(k^2 n)$. The expression Q_0 defines its parse tree, a rooted binary tree whose nodes are the subexpressions of Q_0, and whose edges are the pairs of nodes $\{Q', Q\}$ and $\{Q'', Q\}$ for some subexpression Q of Q_0 with $Q = Q' \oplus_{L,R,S} Q''$ for some L, R, and S.

The algorithm $\mathcal{A}$ works bottom-up in the parse tree T of the slick k-expression Q_0. In other words, it does a depth-first search of the parse tree and operates on the node Q when it returns from the right child Q'' of Q, i.e., it handles Q during the post-visit of Q (i.e., when returning from the search of the last child). At that time also the left child Q' has already been handled.

Let $Q_1, \ldots, Q_{2n-1}$ be the postorder of the vertices of T. This is the order in which the algorithm visits the nodes. To best understand the real algorithm $\mathcal{A}$, one should imagine a virtual algorithm $\tilde{\mathcal{A}}$. It visits the nodes in the same order as the real algorithm. It starts with the matrix $B_0 = B$. In node j, $\tilde{\mathcal{A}}$ does some congruence operation on the matrix B_{j-1} to obtain the matrix B_j. At the end, B_{2n-1} is a diagonal matrix. The matrix B_j is an $n \times n$ matrix. For some j, it might contain $\Omega(n^2)$ nonzero entries, and it might differ from B_{j-1} at $\Omega(n^2)$ places. That's why the real algorithm $\mathcal{A}$ cannot operate directly on the matrices B_j.

Instead, at Q_j the real algorithm $\mathcal{A}$ operates on a small $O(k) \times O(k)$ submatrix. It has to know where this submatrix is located in the full matrix, and it has to know the labels of the currently involved vertices. The algorithm searches through the input expression to simulate a depth-first search through the parse tree of the expression. The algorithm outputs the sequence of diagonal element as it detects them.

Recall that if Q is a slick k-expression which is a subexpression of Q_0, then it is also a node in the parse tree T of Q_0. For a node Q of T, let n_Q be the number of vertices of the graph $G(Q)$ generated by the expression Q. Note that $G(Q)$ is the subgraph of G induced by the vertices of $G(Q)$. Let A_Q be the adjacency matrix of $G(Q)$, and let I_Q be the $n_Q \times n_Q$ identity matrix.

With Q, we associate a matrix $\widetilde{B_Q}$ which is obtained from $B_Q = A_Q - cI_Q$ by some sequence of congruence operations. We will show that such a matrix $\widetilde{B_Q}$ with only $O(k^2)$ nonzero entries can be computed efficiently. Up to simultaneous permutations of rows and columns, $\widetilde{B_Q}$ looks as follows. Of course, such permutations are also congruence operations. Nevertheless, the algorithm does not have to do them, they are just done here to visualize the structure.

$$\widetilde{B_Q} = \begin{pmatrix} \begin{matrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_\ell \end{matrix} & 0 \\ 0 & \begin{array}{|c|c|} \hline M^{(0)} & M^{(1)} \\ \hline M^{(1)\,T} & M^{(2)} \\ \hline \end{array} \end{pmatrix}$$

Here, $M^{(1)}$ is a $k' \times k''$ matrix with $k' \leq k'' \leq k$, where k is the slick clique-width. The partition of the matrix

$$M = \begin{pmatrix} M^{(0)} & M^{(1)} \\ M^{(1)\,T} & M^{(2)} \end{pmatrix}$$

is explained in the next picture. The crucial property of $\widetilde{B_Q}$ is its relationship to the matrix $\widetilde{B_Q^+}$. This is the matrix obtained from B by doing the same congruence operations as for obtaining $\widetilde{B_Q}$ from B_Q. Thus the picture of $\widetilde{B_Q^+}$ also shows how the remainder of the large matrix B is transformed as a side effect of transforming the small submatrix $\widetilde{B_Q}$.

The β_j are some vectors of the original entries of B. It is important to notice the locations of the zero vectors $\mathbf{0}$. In particular, not only is D a diagonal matrix, but all its rows and columns extend with zero vectors. Also the rows and columns of $M^{(0)}$ extend with zero vectors, defining the boundary between $M^{(0)}$ and $M^{(2)}$.

$$\widetilde{B_Q^+} = \begin{pmatrix} D & 0 & 0 & \begin{matrix}\mathbf{0}\\ \vdots \\ \mathbf{0}\end{matrix} \\ 0 & M^{(0)} & M^{(1)} & \begin{matrix}\mathbf{0}\\ \vdots \\ \mathbf{0}\end{matrix} \\ 0 & M^{(1)T} & M^{(2)} & \begin{matrix}\beta_1\\ \vdots \\ \beta_{k''}\end{matrix} \\ \mathbf{0}^T \cdots \mathbf{0}^T & \mathbf{0}^T \cdots \mathbf{0}^T & \beta_1^T \cdots \beta_{k''}^T & M' \end{pmatrix}$$

Before we show how to obtain a matrix of the form $\widetilde{B_Q}$, we show how to enforce some additional properties. We use very simple congruence operations. Besides simultaneous permutations of rows and columns, we only add multiples of some row i to some other row j, followed by adding the same multiple of column i to column j. In other words, we produce P^TBP, where P is a product of matrices which are either permutation matrices or matrices obtained from the unit matrix by adding a single nonzero off diagonal entry.

We say $M^{(0)} = 0$, if we mean $M^{(0)}$ is an all 0 matrix.

Lemma 4. *If the form $\widetilde{B_Q^+}$ can be obtained, then $M^{(0)} = 0$ can be obtained.*

Proof. Let R_i be the i-th row and C_i be the i-th column of M.

If some diagonal element m_{ii} of $M^{(0)}$ is nonzero, then subtract m_{ij}/m_{ii} times the $\ell + i$-th row of $\widetilde{B_Q^+}$ from the $\ell + j$-th row. Do likewise for the columns. In other words, operate on M as follows for all $j \neq i$.

$$R_j \leftarrow R_j - \frac{m_{ij}}{m_{ii}} R_i$$

Note that due to the $\mathbf{0}$ extension of the i-th row, the shape of $\widetilde{B_Q^+}$ does not change, and all the work is restricted to the small matrix M. Now D has one more diagonal element m_{ii}.

If the diagonal of $M^{(0)}$ is 0, but some of its off diagonal elements m_{ij} is nonzero, then do the operations

$$R_j \leftarrow R_j + \frac{1}{2} R_i$$
$$C_j \leftarrow C_j + \frac{1}{2} C_i$$

followed by

$$R_i \leftarrow R_i - R_j$$
$$C_i \leftarrow C_i - C_j.$$

The relevant entries of $M^{(0)}$ transform as follows.

$$\begin{pmatrix} 0 & m_{ij} \\ m_{ij} & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & m_{ij} \\ m_{ij} & m_{ij} \end{pmatrix} \rightarrow \begin{pmatrix} -m_{ij} & 0 \\ 0 & m_{ij} \end{pmatrix}$$

Again, note that due to the surrounding zero pattern, only $M^{(0)}$ and $M^{(1)}$ are modified with the form of $\widetilde{B_Q^+}$ remaining intact. Now the diagonalized part has grown by two. □

Recall that k' and k'' are defined by $M^{(1)}$ being a $k' \times k''$ matrix.

Lemma 5. *If the form $\widetilde{B_Q^+}$ can be obtained with $M^{(0)} = 0$, then $k' \leq k''$ can be enforced.*

Proof. Assume $M^{(0)} = 0$. With simple row operations (subtracting multiples of one row from another one, and doing permutations of rows) $M^{(1)}$ can be made upper triangular. Doing the same operations on the columns of $M^{(1)T}$, the matrix $\widetilde{B_Q^+}$ remains symmetric. If k' was greater than k'', then at least $k' - k''$ rows of $M^{(1)}$ have become 0. Thus the diagonal part of $\widetilde{B_Q^+}$ has grown by $k' - k''$ with zeros in the diagonal. The new k' is at most k'' meaning that $M^{(1)}$ is at most as high as wide. □

Lemma 6. *If the form $\widetilde{B_Q^+}$ can be obtained, then $k'' \leq k$ can be achieved.*

Proof. The β_j are original rows of input matrix B. Consider the labeling of the graph generated by Q. The vertices are labeled with at most k labels. As the labels determine the edges being added later, any two vertices j, j' with the same label have the same neighborhood outside $G(Q)$. This means that $\beta_j = \beta_{j'}$. Thus, if $k'' > k$, then there are at least two such vertices j, j' with $\beta_j = \beta_{j'}$.

Subtracting row j (column j) from row j' (column j' respectively) decreases k'' and increases k' by 1. □

Now we show how to obtain a matrix of the form $\widetilde{B_Q^+}$ from B by a sequence of simple congruence operations. This is proved by induction on the structure of the expression Q, or equivalently on the height of the node Q in the parse tree T of the given expression Q_0.

Lemma 7. *Form $\widetilde{B_Q^+}$ can be obtained by congruence operations from B.*

Proof. If Q is a leaf of T, then Q generates a 1 vertex graph with $B_Q = (-c)$. There is nothing to do, because $\widetilde{B_Q} = Q$ is just fine.

Assume Q is a join node with children Q' and Q'', i.e., $Q = Q' \oplus_{S,L,R} Q''$. We are given $\widetilde{B_{Q'}}$ and $\widetilde{B_{Q''}}$ with the insurance that their production by congruence operations produces $\widetilde{B_{Q'}^+}$ and $\widetilde{B_{Q''}^+}$ of the right forms.

First we form the direct sum of $\widetilde{B_{Q'}}$ and $\widetilde{B_{Q''}}$. We obtain

$$\widetilde{B_Q^{(1)}} = \begin{pmatrix} \widetilde{B_{Q'}} & 0 \\ 0 & \widetilde{B_{Q''}} \end{pmatrix}$$

$\widetilde{B_{Q'}}$ is congruent to $B_{Q'}$ and $\widetilde{B_{Q''}}$ is congruent to $B_{Q''}$. Therefore, $\widetilde{B_Q^{(1)}}$ is congruent to $B_{Q'} \times B_{Q''}$, the adjacency matrix of $Q' \times_{(\emptyset,id,id)} Q''$, where id is the identity function.

This is so, because

$$\widetilde{B_{Q'}} = P'^T B_{Q'} P' \text{ and } \widetilde{B_{Q''}} = P''^T B_{Q'} P''$$

implies

$$\begin{aligned}\begin{pmatrix} \widetilde{B_{Q'}} & 0 \\ 0 & \widetilde{B_{Q''}} \end{pmatrix} &= \begin{pmatrix} P'^T & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P''^T \end{pmatrix} \begin{pmatrix} B_{Q'} & 0 \\ 0 & B_{Q''} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P'' \end{pmatrix} \begin{pmatrix} P' & 0 \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} I & 0 \\ 0 & P''^T \end{pmatrix} \begin{pmatrix} P'^T & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} B_{Q'} & 0 \\ 0 & B_{Q''} \end{pmatrix} \begin{pmatrix} P' & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & P'' \end{pmatrix}\end{aligned}$$

These congruency operators act on independent subspaces. They commute and don't interfere. That's why on the two matrices $B_{Q'}$ and $B_{Q''}$ can be partially diagonalized independently. All areas of the matrix affected by both operations are 0 anyway.

The next step is to introduce the new edges associated with the relation S. As we keep just one vertex, say $u(i)$ of $G(Q')$ labeled i and one vertex, say $v(j)$ of $G(Q'')$ labeled j, we have to just insert a 1 at the two symmetric positions corresponding the vertex $u(i)$ of $G(Q')$ and the vertex $v(j)$ of $G(Q'')$ in the matrix $\widetilde{B_Q^{(1)}}$.

Finally, the vertices are relabeled according to the functions L and R, resulting in $O(k)$ multiple labels. We have already seen in Lemma 6 how to bring k'', down to a maximum of k. During the direct sum operation k'' had temporarily increased to as much as $2k$. □

Theorem 2. *The matrix $B = A - cI$, where A is the adjacency matrix of a graph G (with slick clique-width at most k) given by a slick k-expression Q_0 can be diagonalized in time $O(k^2 n)$.*

Proof. The diagonalization is done bottom-up in the parse tree T of Q_0 according to the previous lemmas. W.l.o.g., we can assume that at the end of the construction all vertices have the same label, as labels are no longer needed. Thus the matrix $\widetilde{B_Q}$ is almost diagonal, because $M^{(0)}, M^{(1)}, M^{(2)}$ are 1×1 matrices. A last step as in Lemma 4 makes it fully diagonal.

The computation can be organized as a depth-first search of the parse tree T. The join operation at node Q is done on postvisit of Q, i.e., when returning from its second child. The interesting part of the data, the matrix M of size $O(k^2)$ can be returned by the recursive calls executing the depth-first search. The diagonal of the already diagonalized part appended to a global array as it is produced. It cannot be passed as a parameter, because this would at least incur a cost of $\Omega(n \log n)$ or even $\Omega(n^2)$ when done carelessly without insuring to copy the smaller piece into the larger.

A time bound of $O(k^3n)$ is straight forward, because the parse tree has $O(n)$ nodes ($2n-1$ to be precise, $n = |V|$ leaves and $n-1$ internal nodes with 2 children each). At each node $O(k^2)$ numbers are handled. But there is an expensive Gaussian elimination happening in the procedure handled by Lemma 5. Here a matrix of size up to $2k$ is put into upper triangular form. This could be handled theoretically in time $k^{\omega+o(1)}$, where ω is the exponent of matrix multiplication, but this is not necessary.

The bound of $O(k^2n)$ is obtained by a better accounting for the time and making sure that no time is wasted in the implementation of Lemma 5. The matrix $M^{(1)}$ is always maintained as an upper triangular matrix. Eliminating one row vector (and thus increasing the diagonalized part by 1) incurs a cost of $O(k^2)$. It does not matter that up to k row vectors are inserted in one node, because every row vector is eliminated only once, and there are only n rows, one for every vertex. □

Having avoided fast matrix multiplication, it is worth stating that the whole algorithm is very fast, as there are no large constants hidden in the O-notation.

Now, we can apply Sylvester's Law of Inertia. Symmetric matrices over the reals have n real eigenvalues (with multiplicities). The inertia of a symmetric real matrix B is the triple (n_+, n_0, n_-) giving the number of eigenvalues of B that are positive, zero, and negative respectively. Sylvester's law says that congruent matrices have the same inertia.

Corollary 1. *The number of eigenvalues of A in a given interval can be computed in time $O(k^2n)$ for graphs of clique-width k.*

Proof. The eigenvalues of $B = A - cI$ are obtained by subtracting c from the eigenvalues of A. To compute the number of eigenvalues of A in the interval $[a, b]$, we run our algorithm with $c = a$ and $c = b$. From the output, we see the numbers of positive, zero, and negative diagonal elements. Let them be (n_+^a, n_0^a, n_-^a) and (n_+^b, n_0^b, n_-^b). Then, obviously the number of eigenvalues in $[a, b]$ is $n_0^b + n_-^b - n_-^a$. □

Remark 2. Another important width parameter is rank-width. With its help, an exponential approximation to the clique-width (and slick clique-width) and the corresponding clique decompositions can be computed by FPT algorithms. Unfortunately, such algorithms are not fast enough yet to serve for our purpose of providing a good clique decomposition. We would want a linear time algorithm.

4 Concluding Remarks

For undirected graphs of bounded clique-width, we are able to diagonalize adjacency matrices with congruence operations extremely efficiently, if a slick clique-width expression of width k or $O(k)$ is given. The algorithm is fairly elegant when either our slick clique-width parameter or the NLC-width is used. In particular, the time is much faster than the time to read an explicitly given matrix.

It is worth stating that the whole algorithm is very fast, as there are no large constants hidden in the O-notation.

A main question is about extensions to other width parameters, as well as the computation of other related matrix tasks. It would be interesting if theoretical results about the location of eigenvalues for special classes of graphs could be derived from analyzing the behavior of our algorithm on these graphs, as it has been done for cographs [10].

References

1. Alazemi, A., Andelić, M., Simić, S.K.: Eigenvalue location for chain graphs. Linear Algebra Appl. **505**, 194–210 (2016)
2. Bıyıkoğlu, T., Simić, S.K., Stanić, Z.: Some notes on spectra of cographs. Ars Combin. **100**, 421–434 (2011)
3. Braga, R.O., Rodrigues, V.M., Trevisan, V.: Locating eigenvalues of unicyclic graphs. Appl. Anal. Discrete Math. **11**(2), 273–298 (2017)
4. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, pp. 313–400. World Scientific (1997)
5. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. J. Comput. Syst. Sci. **46**(2), 218–270 (1993)
6. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Appl. Math. **101**(1–3), 77–114 (2000)
7. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width minimization is NP-hard (extended abstract). In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 354–362. ACM, New York (2006)
8. Jacobs, D.P., Trevisan, V.: Locating the eigenvalues of trees. Linear Algebra Appl. **434**(1), 81–88 (2011)
9. Jacobs, D.P., Trevisan, V., Tura, F.: Eigenvalue location in threshold graphs. Linear Algebra Appl. **439**(10), 2762–2773 (2013)
10. Jacobs, D.P., Trevisan, V., Tura, F.C.: Eigenvalue location in cographs. Discrete Appl. Math. (2017)
11. Janssens, D., Rozenberg, G.: On the structure of node-label-controlled graph languages. Inf. Sci. **20**(3), 191–216 (1980)
12. Janssens, D., Rozenberg, G.: Restrictions, extensions, and variations of NLC grammars. Inf. Sci. **20**(3), 217–244 (1980)
13. Johansson, Ö.: Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. Congr. Numer. **132**, 39–60 (1998)
14. Kaminski, M., Lozin, V.V., Milanic, M.: Recent developments on graphs of bounded clique-width. Discrete Appl. Math. **157**(12), 2747–2761 (2009)
15. Meyer, C.: Matrix Analysis and Applied Linear Algebra. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2000). With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.)
16. Mohammadian, A., Trevisan, V.: Some spectral properties of cographs. Discrete Math. **339**(4), 1261–1264 (2016)

17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
18. Robertson, N., Seymour, P.D.: Graph minors II. Algorithmic aspects of tree-width. J. Algorithms **7**(3), 309–322 (1986)
19. Royle, G.F.: The rank of a cograph. Electron. J. Combin. **10**, Note 11, 7 pp. (electronic) (2003)
20. Sander, T.: On certain eigenspaces of cographs. Electron. J. Comb. **15**(1), 8 (2008)
21. Stanić, Z.: On nested split graphs whose second largest eigenvalue is less than 1. Linear Algebra Appl. **430**(8–9), 2200–2211 (2009)
22. Wanke, E.: k-NLC graphs and polynomial algorithms. Discrete Appl. Math. **54**(2), 251–266 (1994)

On the Approximation Ratio of Lempel-Ziv Parsing

Travis Gagie[1,2], Gonzalo Navarro[2,3](✉), and Nicola Prezza[4]

[1] EIT, Diego Portales University, Santiago, Chile
[2] Center for Biotechnology and Bioengineering (CeBiB), Santiago, Chile
gnavarro@dcc.uchile.cl
[3] Department of Computer Science, University of Chile, Santiago, Chile
[4] DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

Abstract. Shannon's entropy is a clear lower bound for statistical compression. The situation is not so well understood for dictionary-based compression. A plausible lower bound is b, the least number of phrases of a general bidirectional parse of a text, where phrases can be copied from anywhere else in the text. Since computing b is NP-complete, a popular gold standard is z, the number of phrases in the Lempel-Ziv parse of the text, where phrases can be copied only from the left. While z can be computed in linear time, almost nothing has been known for decades about its approximation ratio with respect to b. In this paper we prove that $z = O(b \log(n/b))$, where n is the text length. We also show that the bound is tight as a function of n, by exhibiting a string family where $z = \Omega(b \log n)$. Our upper bound is obtained by building a run-length context-free grammar based on a locally consistent parsing of the text. Our lower bound is obtained by relating b with r, the number of equal-letter runs in the Burrows-Wheeler transform of the text. On our way, we prove other relevant bounds between compressibility measures.

1 Introduction

Shannon [33] defined a measure of entropy that serves as a lower bound to the attainable compression ratio on any source that emits symbols according to a certain probabilistic model. An attempt to measure the compressibility of finite texts $T[1 \ldots n]$, other than the non-computable Kolmogorov complexity [21], is the notion of empirical entropy [7], where some probabilistic model is assumed and its parameters are estimated from the frequencies observed in the text. Other measures that, if the text is generated from a probabilistic source, converge to its Shannon entropy, are derived from the Lempel-Ziv parsing [23] or the grammar-compression [20] of the text.

Some text families, however, are not well modeled as coming from a probabilistic source. A very current case is that of *highly repetitive texts*, where most

Partially funded by Basal Funds FB0001, Conicyt, by Fondecyt Grants 1-171058 and 1-170048, Chile, and by the Danish Research Council DFF-4005-00267.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 490–503, 2018.
https://doi.org/10.1007/978-3-319-77404-6_36

of the text can be obtained by copying long blocks from elsewhere in the same text. Huge highly repetitive text collections are arising from the sequencing of myriads of genomes of the same species, from versioned document repositories like Wikipedia, from source code repositories like GitHub, etc. Their growth is outpacing Moore's Law by a wide margin [34]. Understanding the compressibility of highly repetitive texts is important to properly compress those huge collections.

Lempel-Ziv and grammar compression are particular cases of so-called *dictionary techniques*, where a set of strings is defined and the text is parsed as a concatenation of those strings. On repetitive collections, the empirical entropy ceases to be a relevant compressibility measure; for example the kth order per-symbol entropy of TT is the same as that of T, if $k \ll n$ [22, Lemma 2.6], whereas this entropy measure is generally meaningless for $k > \log n$ [12]. Some dictionary measures, instead, capture much better the compressibility of repetitive texts. For example, while an individual genome can rarely be compressed to much less than 2 bits per symbol, Lempel-Ziv has been reported to compress collections of human genomes to less than 1% [11]. Similar compression ratios are reported in Wikipedia.[1]

Despite the obvious practical relevance of these compressibility measures, there is not a clear entropy measure useful for highly repetitive texts. The number z of phrases generated by the Lempel-Ziv parse [23] is often used as a gold standard, possibly because it can be implemented in linear time [30] and is never larger than g, the size of the smallest context-free grammar that generates the text [6,31]. However, z is not so satisfactory as an entropy measure: the value changes if we reverse the text, for example. A much more robust lower bound on compressibility is b, the size of the smallest *bidirectional (macro) scheme* [35]. Such a scheme parses the text into phrases such that each phrase appears somewhere else in the text (or it is a single explicit symbol), so that it is possible to recover the text by copying source to target positions in an appropriate order. This is arguably the strongest possible dictionary method, but finding the smallest bidirectional scheme is NP-complete [13]. A relevant question is then how good is the Lempel-Ziv parse as an efficiently implementable approximation to the smallest bidirectional scheme. Almost nothing is known in this respect, except that there are string families where z is nearly $2b$ [35].

In this paper we finally give a tight approximation ratio for z, showing that the gap is larger than what was previously known. We prove that $z = O(b \log(n/b))$, and that this bound is tight as a function of n, by exhibiting a string family where $z = \Omega(b \log n)$. To prove the upper bound, we show how to build a run-length context-free grammar [28] (i.e., allowing rules of the form $X \rightarrow Y^t$ that count as size 1) of size $g_{rl} = O(b \log(n/b))$. This is done by carrying out several rounds of locally consistent parsing [17] on top of T, reducing the resulting blocks to nonterminals in each round, and showing that new nonterminals appear only in the boundaries of the phrases of the bidirectional scheme. We then further prove that $z \leq 2g_{rl}$, by extending a classical proof [6] that relates

[1] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia.

grammar with Lempel-Ziv compression. To prove the lower bound, we consider another plausible compressibility measure: the number r of equal-symbol runs in the Burrows-Wheeler transform (BWT) of the text [5]. We prove that the BWT induces a valid bidirectional scheme, and thus $r = \Omega(b)$. Then the bound follows from known string families where $z = \Omega(r \log n)$ [29].

2 Basic Concepts

A string is a sequence $S[1 \dots \ell] = S[1]S[2] \dots S[\ell]$ of symbols. A substring $S[i] \dots S[j]$ of S is denoted $S[i \dots j]$. A suffix of S is a substring of the form $S[i \dots \ell]$. The juxtaposition of strings and/or symbols represents their concatenation. We will consider compressing a string $T[1 \dots n]$, called the *text*.

2.1 Bidirectional Schemes

A bidirectional scheme [35] partitions $T[1 \dots n]$ into b chunks $B_1, \dots, B_b$, such that each $B_i = T[t_i \dots t_i + \ell_i - 1]$ (called a target) is either (1) copied from another substring $T[s_i \dots s_i + \ell_i - 1]$ (called a source) with $s_i \neq t_i$, which may overlap $T[t_i \dots t_i + \ell_i - 1]$, or (2) formed by $\ell_i = 1$ explicit symbol.

We define the function $f : [1 \dots n] \rightarrow [1 \dots n]$ so that, in case (1), $f(t_i + j) = s_i + j$ for all $0 \leq j < \ell_i$, and in case (2), $f(t_i) = -1$. Then, the bidirectional scheme is valid if there is an order in which the sources $s_i + j$ can be copied onto the targets $t_i + j$ so that all the positions of T can be inferred.

Being a valid scheme is equivalent to saying that f has no cycles, that is, there is no $k > 0$ and p such that $f^k(p) = p$: Initially we can set all the explicit positions (type (2)), and then copy sources with known values to their targets. If f has no cycles, we will eventually complete all the positions in T because, for every $T[p]$, there is a $k > 0$ such that $f^k(p) = -1$, so we can obtain $T[p]$ from the symbol explicitly stored for $T[f^{k-1}(p)]$.

We use b to denote the smallest bidirectional scheme, which is NP-complete to compute [13].

2.2 Lempel-Ziv Parsing

Lempel and Ziv [23] define a parsing of T into the fewest possible phrases $T = Z_1 \dots Z_z$, so that each phrase Z_i is a substring (but not a suffix) of $Z_1 \dots Z_i$, or a single symbol. This means that the source $T[s_i \dots s_i + \ell_i - 1]$ of the target $Z_i = T[t_i \dots t_i + \ell_i - 1]$ must satisfy $s_i < t_i$, but sources and targets may overlap. It turns out that the greedy left-to-right parsing indeed produces the optimal number z of phrases [23, Theorem 1]. Further, the parsing can be obtained in $O(n)$ time [30,35].

If we disallow that a phrase overlaps its source, that is, Z_i must be a substring of $Z_1 \dots Z_{i-1}$ or a single symbol, then we call z_{no} the number of phrases obtained. In this case it is also true that the greedy left-to-right parsing produces the optimal number z_{no} of phrases [35, Theorem 10 with $p = 1$]. Since the Lempel-Ziv parsing allowing overlaps is optimal among all left-to-right parsings, we also have that $z_{no} \geq z$. This parsing can also be computed in $O(n)$ time [8]. Note that, on a text family like $T = a^n$, it holds that $z_{no} = \Omega(z \log n)$.

Little is known about the relation between b and z except that $z \geq b$ by definition (z is the smallest *unidirectional* parsing) and that, for any constant $\epsilon > 0$, there is an infinite family of strings for which $b < (\frac{1}{2} + \epsilon) \cdot \min(z, z^R)$ [35, Correlation 7.1], where z^R is the z of the reversed string.

2.3 Grammar Compression

Consider a context-free grammar (CFG) that generates T and only T [20]. Each nonterminal must be the left-hand side in exactly one rule, and the size g of the grammar is the sum of the right-hand sides of the rules. In general, we will use g to denote the minimum possible size of a grammar that generates T, which is NP-complete to compute [6,31].

If we allow, in addition, rules of the form $X \rightarrow Y^t$, of size 1, the result is a run-length context-free grammar (RLCFG) [28]. We will use g_{rl} to denote the size of the smallest RLCFG that generates T. Thus, it is clear that $g_{rl} \leq g$. Further, on the string family $T = a^n$ it holds that $g = \Omega(g_{rl} \log n)$.

A well-known relation between z_{no} and g is $z_{no} \leq g = O(z_{no} \log(n/z_{no}))$ [6,31]. Further, it is known that $g = O(z \log(n/z))$ [14, Lemma 8]. Those papers exhibit $O(\log n)$-approximations to the smallest grammar, as well as several others [17,18,32]. A negative result about the approximation are string families where $g = \Omega(z_{no} \log n / \log \log n)$ [6,15] and, recently, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [3].

2.4 Runs in the Burrows-Wheeler Transform

Assume that T is terminated by the special symbol $T[n] = \$$, which is lexicographically smaller than all the others. This makes any lexicographic comparison between suffixes well defined.

The *suffix array* [25] of $T[1 \ldots n]$ is an array $SA[1 \ldots n]$ storing a permutation of $[1 \ldots n]$ so that, for all $1 \leq i < n$, the suffix $T[SA[i] \ldots]$ is lexicographically smaller than the suffix $T[SA[i+1] \ldots]$. Thus $SA[i]$ is the starting position in T of the ith smallest suffix of T in lexicographic order.

The inverse permutation of SA, $ISA[1 \ldots n]$, is called the *inverse suffix array*, so that $ISA[j]$ is the lexicographical position of the suffix $T[j \ldots n]$ among the suffixes of T.

The *Burrows-Wheeler Transform* of $T[1 \ldots n]$, $BWT[1 \ldots n]$ [5], is a string defined as $BWT[i] = T[SA[i] - 1]$ if $SA[i] > 1$, and $BWT[i] = T[n] = \$$ if $SA[i] = 1$. That is, BWT has the same symbols of T in a different order, and is a reversible transform.

The array BWT can be easily obtained from T and SA, which can be built in $O(n)$ time [19]. To obtain T from BWT [5], one considers two arrays, $L[1 \ldots n] =$

BWT and $F[1 \dots n]$, which contains all the symbols of L (or T) in ascending order. Alternatively, $F[i] = T[SA[i]]$, so $F[i]$ follows $L[i]$ in T. We need a function that maps any $L[i]$ to the position j of that same symbol in F. The formula is $LF(i) = C[c] + \text{rank}[i]$, where $c = L[i]$, $C[c]$ is the number of occurrences of symbols less than c in L, and rank$[i]$ is the number of occurrences of symbol $L[i]$ in $L[1 \dots i]$. Once C and rank are computed, we reconstruct $T[n] = \$$ and $T[n-k] \leftarrow L[LF^{k-1}(1)]$ for $k = 1, \dots, n-1$.

The number of equal-symbol runs r in the BWT of T can be bounded in terms of the empirical entropy [24]. However, the measure is also interesting on highly repetitive collections (where, in particular, z and z_{no} are small). For example, there are string families where $z = \Omega(r \log n)$ [29], and others where $r = \Omega(z_{no} \log n)$ [2,29].

2.5 Locally Consistent Parsing

A string can be parsed in a *locally consistent* way, in the sense that equal substrings are largely parsed in the same form. We use a variant of locally consistent parsing called recompression [16,17].

Definition 1. *A* repetitive area *in a string is a maximal run of the same symbol, of length 2 or more.*

Definition 2. *Two segments contained in* $[1 \dots n]$ overlap *if they are not disjoint nor one contained in the other.*

Lemma 1 ([17]). *We can partition a string* $S[1 \dots \ell]$ *into at most* $(3/4)\ell$ *blocks so that, for every pair of identical substrings* $S[i \dots j] = S[i' \dots j']$, *if neither* $S[i+1 \dots j-1]$ *or* $S[i'+1 \dots j'-1]$ *overlap a repetitive area, then the sequence of blocks covering* $S[i+1 \dots j-1]$ *and* $S[i'+1 \dots j'-1]$ *are identical.*

Proof. The parsing is obtained by, first, creating new symbols that represent the repetitive areas. On the resulting sequence, the alphabet (which contains original symbols and created ones) is partitioned into two subsets, left-symbols and right-symbols. Then, every left-symbol followed by a right-symbol are paired in a block. It is then clear that, if $S[i+1 \dots j-1]$ and $S[i'+1 \dots j'-1]$ do not overlap repetitive areas, then the parsing of $S[i \dots j]$ and $S[i' \dots j']$ may differ only in their first position (if it is part of a repetitive area ending there, or if it is a right-symbol that becomes paired with the preceding one) and in their last position (if it is part of a repetitive area starting there, or if it is a left-symbol that becomes paired with the following one). Jez [17] shows how to choose the pairs so that S contains at most $(3/4)\ell$ blocks. $\square$

The lemma ensures a locally consistent parsing into blocks as long as the substrings do not overlap repetitive areas, though the substrings may fully contain repetitive areas.

3 Upper Bounds

In this section we obtain our main upper bound, $z = O(b\log(n/b))$, along with other byproducts. To this end, we first prove that $g_{rl} = O(b\log(n/b))$, and then that $z \leq 2g_{rl}$. To prove the first bound, we build a RLCFG on top of a bidirectional scheme. The grammar is built in several rounds of locally consistent parsing on the text. In each round, the blocks of the locally consistent parsing are converted into nonterminals and fed to the next round. The key is to prove that distinct nonterminals are produced only at the boundaries of the phrases of the bidirectional scheme. The second bound is an easy extension to the known result $z_{no} \leq g$.

Theorem 1. *Let $T[1\ldots n]$ have a bidirectional scheme of size b. Then there exists a run-length context-free grammar of size $g_{rl} = O(b\log(n/b))$ that generates T.*

Proof. Recalling Lemma 1, consider a locally consistent parsing of $W = T$ into blocks. We will count the number of *different* blocks we form, as this corresponds to the number of nonterminals produced in the first round.

Recall from Sect. 2.1 that our bidirectional scheme represents T as a sequence of *chunks*, by means of a function f. To count the number of different blocks produced, we will pessimistically assume that the first two and the last two blocks intersecting each chunk are all different. The number of such *bordering* blocks is at most $4b$. On the other hand, we will show that non-bordering blocks do not need to be considered, because they will be counted somewhere else, when they appear near the extreme of a chunk.

We show that this is true in both types of non-bordering blocks resulting from Lemma 1:

1. The block is a pair of left- and right-alphabet symbols.[2] As these symbols can be an original symbol or a maximal area, let us write the pair generically as $X = a^{\ell_a}b^{\ell_b}$, for some $\ell_a, \ell_b \geq 1$, and let $\ell = \ell_a + \ell_b$ be the length of the block X. If $W[p\ldots p+\ell-1] = X$ is not bordering, then it is strictly contained in a chunk. Thus, by the definition of a chunk, it holds that $[f(p-1)\ldots f(p+\ell)] = [f(p)-1\ldots f(p)+\ell]$, and that $W[f(p)-1\ldots f(p)+\ell] = W[p-1\ldots p+\ell]$. That is, the block appears again at $[f(p)\ldots f(p)+\ell-1]$, surrounded by the same symbols. Since, by the way Lemma 1 works, it must be $W[f(p)-1] = W[p-1] \neq a$ and $W[f(p)+\ell] = W[p+\ell] \neq b$, and a^{ℓ_a} is a left-symbol and b^{ℓ_b} is a right-symbol, the locally consistent parsing must also form a block $W[f(p)\ldots f(p)+\ell-1] = X$. If this block is bordering, then it will be counted. Otherwise, by the same argument, $W[f(p)-1\ldots f(p)+\ell]$ will be equal to $W[f^2(p)-1\ldots f^2(p)+\ell]$ and a block will be formed with $W[f^2(p)\ldots f^2(p)+\ell-1]$. Since f has no cycles, there is a $k > 0$ for which $f^k(p) = -1$. Thus for some $l < k$ it must be that $X = W[f^l(p)\ldots f^l(p)+\ell-1]$

[2] For this case, we could have defined *bordering* in a stricter way, as the first or last block of a chunk.

is not bordering. At the smallest such l, the block $W[f^l(p) \dots f^l(p)+\ell-1]$ will be counted. Therefore, $X = W[p \dots p+\ell-1]$ is already counted somewhere else and we do not need to count it at $W[p \dots p+\ell-1]$.

2. The block is a single (original or maximal-run) symbol $W[p \dots p+\ell-1] = a^\ell$, for some $\ell \geq 1$. It also holds that $[f(p-1) \dots f(p+\ell)] = [f(p)-1 \dots f(p)+\ell]$ and $W[f(p)-1..f(p)+\ell] = W[p-1 \dots p+\ell]$, because a^ℓ is strictly inside a chunk. Since $W[f(p)-1] = W[p-1] \neq a$ and $W[f(p)+\ell] = W[p+\ell] \neq a$, the parsing forms the same maximal run $a^\ell = W[f(p) \dots f(p)+\ell-1]$. Moreover, since $W[p \dots p+\ell-1]$ is not bordering, the previous and next blocks produced by the parsing, $X = W[p' \dots p-1]$ and $Y = [p+\ell \dots p'']$, are also strictly inside the same chunk, and therefore they also appear preceding and following $W[f(p) \dots f(p)+\ell-1]$, at $X = W[f(p') \dots f(p)-1]$ and $Y = [f(p)+\ell \dots f(p'')]$. Since a^ℓ was not paired with X nor Y at $W[p \dots p+\ell-1]$, the parsing will also not pair them at $W[f(p) \dots f(p)+\ell-1]$. Therefore, the parsing will leave a^ℓ as a block also in $[f(p) \dots f(p)+\ell-1]$. If $W[f(p) \dots f(p+\ell-1)]$ is bordering, then it will be counted, otherwise we can repeat the argument with $W[f^2(p)-1 \dots f^2(p)+\ell]$ and so on, as in the previous item.

Therefore, we produce at most $4b$ distinct blocks, and the RLCFG has at most $12b$ nonterminals (for $X = a^{\ell_a} b^{\ell_b}$ we may need 3 nonterminals, $A \rightarrow a^{\ell_a}$, $B \rightarrow b^{\ell_b}$, and $C \rightarrow AB$).

For the second round, we create a reduced sequence W' from W by replacing all the blocks of length 2 or more by their corresponding nonterminals. The new sequence is guaranteed to have length at most $(3/4)n$ by Lemma 1.

We define a new bidirectional scheme (recall Sect. 2.1) on W', as follows:

1. For each bordering block in W, its nonterminal symbol position in W' is made explicit in the bidirectional scheme of W'. Note that this includes the blocks covering the explicit symbols in the bidirectional scheme of W.
2. For the chunks $B_i = W[t_i \dots t_i+\ell_i-1]$ of W containing non-bordering blocks (note B_i cannot be an explicit chunk), let B'_i be obtained by trimming from B_i the bordering blocks near the extremes of B_i. Then B'_i appears inside $W[s_i \dots s_i+\ell_i-1]$ (with $s_i = f(t_i)$), where the same sequence of blocks is formed by our arguments above. We then form a chunk in W' with sequence of nonterminals associated with the blocks of B'_i (all of which are non-bordering), pointing to the identical sequence of nonterminals that appear as blocks inside $W[s_i \dots s_i+\ell_i-1]$.

To bound the total number of nonterminals generated, let us call W_k the sequence W after k iterations (so $T = W_0$) and N_k the number of distinct blocks created when converting W_k into W_{k+1}.

In the first iteration, since there may be up to 4 bordering blocks around each chunk limit, we may create $N_1 \leq 4b$ distinct blocks. Those blocks become new explicit chunks in the bidirectional scheme of $W' = W_1$. Note that those explicit chunks are grouped into b *regions* of up to 4 consecutive chunks. In each new iteration, W_k is parsed into blocks again. We have shown that the blocks formed outside regions (i.e., non-bordering blocks) are not distinct, so we can focus on the number of new blocks produced to parse each of the b regions. The parsing produces at most 4 new distinct blocks extending each region. However, the parsing of the regions themselves may also produce new distinct blocks. Our aim is to show that the number of those blocks is also bounded because they decrease the length of the regions, which only grow by $4b$ per iteration.

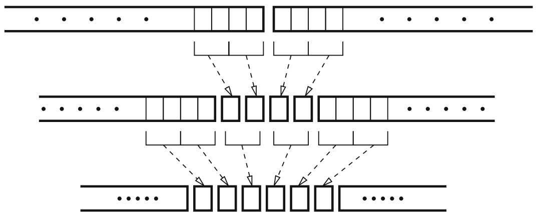


Fig. 1. Illustration of Theorem 1. On top we see the limit between two long chunks of W_0. In this example, the blocking always pairs two symbols. We show below W_0 the 4 bordering blocks formed with the symbols nearby the limit. Below, in W_1, those blocks are converted into 4 explicit chunks (of length 1). This region of 4 symbols is then parsed into 2 blocks. The parsing also creates 4 new bordering blocks from the ends of the long chunks. In W_2, below, we have now a region of 6 explicit chunks. They could have been 8, but we created 2 distinct blocks that reduced their number to 6.

Let n_k be the number of new distinct blocks produced when parsing the regions themselves. Therefore it holds that the number of distinct blocks N_k produced in the kth iteration is at most $4b+n_k$, and the total number of distinct blocks created up to building W_k is $\sum_{i=0}^{k-1} N_i \leq 4bk + \sum_{i=0}^{k-1} n_i$.

On the other hand, for each of the n_k blocks created when parsing a region, the length of the region decreases at least by 1 in W_{k+1}. Let us call C_k the number of explicit chunks in W_k. Since only the 4 new bordering blocks at each region are converted into explicit chunks, it holds that $C_k \leq 4bk$ for all $k > 0$. Moreover, it holds $C_{k+1} \leq C_k + 4b - n_k$, and thus $0 \leq C_k \leq 4bk - \sum_{i=0}^{k-1} n_i$. Therefore, $\sum_{i=0}^{k-1} n_i \leq 4bk$ and thus $\sum_{i=0}^{k-1} N_i \leq 8bk$. Since each nonterminal may need 3 rules to represent a block, a bound on the number of nonterminals created is $24bk$.

After k rounds, the sequence is of length at most $(3/4)^k n$ and we have generated at most $24bk$ nonterminals. Therefore, if we choose to perform $k = \log_{4/3}(n/b)$ rounds, the sequence will be of length at most b and the grammar size will be $O(b\log(n/b))$. To complete the process, we add $O(b)$ nonterminals to reduce the sequence to a single initial symbol.

The idea is illustrated in Fig. 1. $\square$

With Theorem 1, we can also bound the size z of the Lempel-Ziv parse [23] that allows overlaps. The size without allowing overlaps is known to be bounded by the size of the smallest CFG, $z_{no} \leq g$ [6,31]. We can easily see that $z \leq 2g_{rl}$ also holds by extending an existing proof [6, Lemma 9] to handle the run-length rules. We call *left-to-right parse* of T any parsing in which each new phrase is a symbol or it occurs previously in T.

Theorem 2. *Let a RLCFG of size g_{rl} expand to a text T. Then the Lempel-Ziv parse (allowing overlaps) of T produces $z \leq 2g_{rl}$ phrases.*

Proof. Consider the parse tree of T, where all internal nodes representing any but the leftmost occurrence of a nonterminal are pruned and left as leaves. The number of nodes in this tree is precisely g_{rl}. We say that the internal node of nonterminal X is its definition. Our left-to-right parse of T is a sequence $Z[1 \dots z]$ obtained by traversing the leaves of the pruned parse tree left to right. For a terminal leaf, we append the symbol to Z. For a leaf representing nonterminal X, we append to Z a reference to the area $T[x \dots y]$ expanded by the leftmost occurrence of X.

Rules $X \rightarrow Y^t$ are handled as follows. First, we expand them to $X \rightarrow Y \cdot Y^{t-1}$, that is, the node for X has two children for Y, and it is annotated with $t-1$. Since the right child of X is not the first occurrence of Y, it must be a leaf. The left child of X may or may not be a leaf, depending on whether Y occurred before or not. Now, when our leaf traversal reaches the right child Y of a node X indicating $t-1$ repetitions, we append to Z a reference to $T[x \dots y + (t-2)(y-x+1)]$, where $T[x \dots y]$ is the area expanded by the first child of X. Note that source and target overlap if $t > 2$. Thus a left-to-right parse of size $2g_{rl}$ exists, and Lempel-Ziv is the optimal left-to-right parse [23, Theorem 1]. □

By combining Theorems 1 and 2, we obtain a result on the long-standing open problem of finding the approximation ratio of Lempel-Ziv compared to the smallest bidirectional scheme.

Theorem 3. *Let $T[1 \dots n]$ have a bidirectional scheme of size b. Then the Lempel-Ziv parsing of T allowing overlaps has $z = O(b \log(n/b))$ phrases.*

We can also derive upper bounds for g, the size of the smallest CFG, and for z_{no}, the size of the Lempel-Ziv parse that does not allow overlaps. It is sufficient to combine the previous results with the facts that $g = O(z \log(n/z))$ [14, Lemma 8] and $z_{no} \leq g$ [6,31].

Theorem 4. *Let $T[1 \dots n]$ have a bidirectional scheme of size b. Then there exists a context-free grammar of size $g = O(b \log^2(n/b))$ that generates T.*

Theorem 5. *Let $T[1 \dots n]$ have a bidirectional scheme of size b. Then the Lempel-Ziv parsing of T without allowing overlaps has $z_{no} = O(b \log^2(n/b))$ phrases.*

4 Lower Bounds

In this section we prove that the upper bound of Theorem 3 is tight as a function of n, by exhibiting a family of strings for which $z = \Omega(b \log n)$. This confirms that

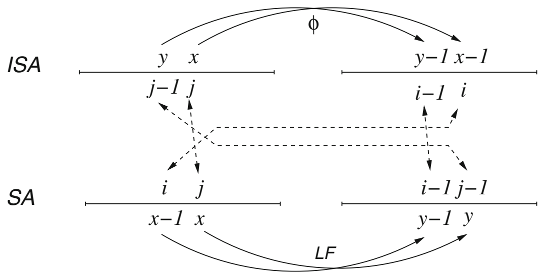


Fig. 2. Illustration of Lemma 2.

the gap between bidirectionality and unidirectionality is significantly larger than what was previously known. The idea is to define phrases in T accordingly to the r runs in the BWT, and to show that these phrases induce a valid bidirectional macro scheme of size $2r$. This proves that $r = \Omega(b)$. Then we use a well-known family of strings where $z = \Omega(r \log n)$.

Definition 3. *Let $p_1, p_2, \ldots, p_r$ be the positions that start runs in the BWT, and let $s_1 < s_2 < \ldots < s_r$ be the corresponding positions in T, $\{SA[p_i], 1 \le i \le r\}$, in increasing order. Note that $s_1 = 1$ because $BWT[ISA[1]] = \$$ is a size-1 run, and assume $s_{r+1} = n+1$, so that T is partitioned into* phrases $T[s_i \ldots s_{i+1}-1]$. *Let also $\phi(i) = SA[ISA[i]-1]$ if $ISA[i] > 1$ and $\phi(i) = SA[n]$ otherwise. Then we define the* bidirectional scheme of the BWT:

1. *For each $1 \le i \le r$, $T[\phi(s_i) \ldots \phi(s_{i+1}-2)]$ is copied from $T[s_i \ldots s_{i+1}-2]$.*
2. *For each $1 \le i \le r$, $T[\phi(s_{i+1}-1)]$ is stored explicitly.*

We build on the following lemma, illustrated in Fig. 2.

Lemma 2. *Let $[j-1 \ldots j]$ be within a phrase of T. Then it holds that $\phi(j-1) = \phi(j)-1$ and $T[j-1] = T[\phi(j)-1]$.*

Proof. Consider the pair of positions $T[j-1 \ldots j]$ within a phrase. Let them be pointed from $SA[x] = j$ and $SA[y] = j-1$, therefore $ISA[j] = x$, $ISA[j-1] = y$, and $LF(x) = y$. Now, since j is not a position at the beginning of a phrase, x is not the first position in a BWT run. Therefore, $BWT[x-1] = BWT[x]$, from which it follows that $LF(x-1) = LF(x)-1 = y-1$. Now let $SA[x-1] = i$, that is, $i = \phi(j)$. Then $\phi(j-1) = SA[ISA[j-1]-1] = SA[y-1] = SA[LF(x-1)] = SA[x-1]-1 = i-1 = \phi(j)-1$. It also follows that $T[j-1] = BWT[x] = BWT[x-1] = T[i-1] = T[\phi(j)-1]$. □

Lemma 3. *The bidirectional scheme of the BWT is a valid bidirectional scheme, thus $2r \ge b$.*

Proof. By Lemma 2, it holds that $\phi(j-1) = \phi(j)-1$ if $[j-1 \ldots j]$ is within a phrase, and that $T[j-1] = T[\phi(j)-1]$. Therefore, we have that $\phi(s_i+k) = \phi(s_i)+k$ for $0 \le k < s_{i+1}-s_i-1$, and then $T[\phi(s_i), \ldots, \phi(s_{i+1}-2)]$ is indeed a

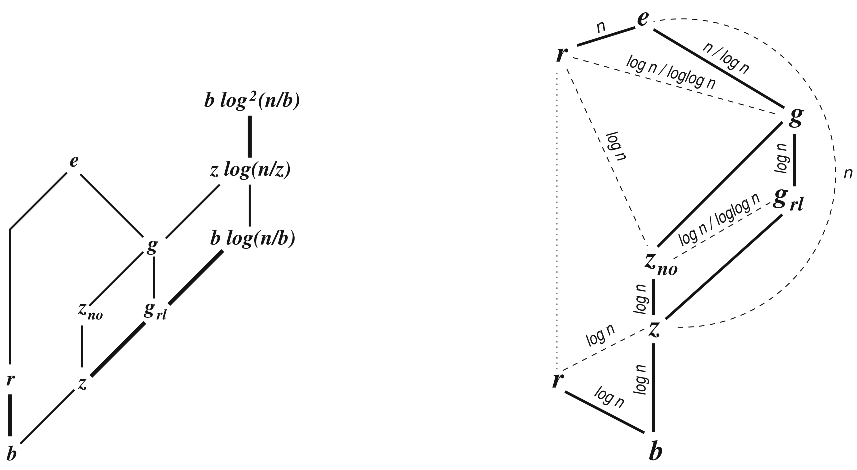


Fig. 3. Known and new asymptotic bounds between repetitiveness measures. The bounds on the left hold for every string family: an edge means that the lower measure is of the order of the upper. The thicker lines were proved in this paper. The dashed lines on the right are lower bounds that hold for some string family. The solid lines are inherited from the left, and since they always hold, they permit propagating the lower bounds. Note that r appears twice.

contiguous range. We also have that $T[\phi(s_i)\dots\phi(s_{i+1}-2)] = T[s_i \dots s_{i+1}-2]$, and therefore it is correct to make the copy. Since ϕ is a permutation, every position of T is mentioned exactly once as a target in points 1 and 2.

Finally, it is easy to see that we can recover the whole T from those $2r$ directives. We can, for example, follow the cycle $\phi^k(n)$, $k = 0, \dots, n-1$ (note that $T[\phi^0(n)] = T[n]$ is stored explicitly), and copy $T[\phi^k(n)]$ to $T[\phi^{k+1}(n)]$ unless the latter is explicitly stored.

Since the bidirectional scheme of the BWT is of size $2r$, it follows by definition that $2r \geq b$. $\square$

We are now ready to obtain the lower bound on bidirectional versus unidirectional parsings.

Theorem 6. *There is an infinite family of strings over an alphabet of size 2 for which* $z = \Omega(b \log n)$.

Proof. Consider the family of the Fibonacci stings, $F_1 = a$, $F_2 = b$, and $F_k = F_{k-1}F_{k-2}$ for all $k > 2$. As observed by Prezza [29, Theorem 25], for F_k we have $r = O(1)$ [26] and $z = \Theta(\log n)$ [10]. By Lemma 3, it also holds that $b = O(1)$, and therefore $z = \Omega(b \log n)$. $\square$

5 Conclusions

We have essentially closed the question of which is the approximation ratio of the (unidirectional) Lempel-Ziv parse with respect to the optimal bidirectional parse, therefore contributing to the understanding of the quality of this popular heuristic that can be computed in linear time, whereas computing the optimal bidirectional parse is NP-complete. Our bounds, which are shown to be tight, show that the gap is in fact wider than what was previously known.

Figure 3(left) illustrates the known asymptotic bounds that relate the repetitiveness measures we have studied: b, z, z_{no}, g, g_{rl}, and r. We also include e, the size of the CDAWG [4] of T (i.e., the smallest compact automaton that recognizes the substrings of T), which has received some attention recently [2]. It is known that $e \geq \max(z, r)$ [2] and $e = \Omega(g)$ [1].

Figure 3(right) shows known lower bounds that hold for specific string families. Apart from the lower bounds mentioned in Sect. 2, there are text families for which $e = \Omega(\max(r, z) \cdot n)$ [2] and thus $e = \Omega(g \cdot n/\log n)$ since $g = O(z \log n)$; and $r = \Omega(g \log n/\log\log n)$ (since on a de Bruijn sequence of order k on a binary alphabet we have $r = \Theta(n)$ [2], $z = O(n/\log n)$, and thus $g = O(z \log(n/z)) = O(n \log\log n/\log n)$). From the upper bounds that hold for every string family, we can also deduce that, for example, there are string families where $r = \Omega(z \log n)$ and thus $r = \Omega(b \log n)$ (since $r = \Omega(z_{no} \log n)$); $\{g, g_{rl}, z_{no}\} = \Omega(r \log n)$ (since $z = \Omega(r \log n)$) and $z = \Omega(b \log n)$ (since $r = \Omega(b)$, Theorem 6). We nevertheless included explicitly the most important of these in the figure.

There are various interesting avenues of future work. For example, it is unknown if r can be more than $O(\log n)$ times larger than z or g. It might also be that our Theorem 1 can be proved without using run-length rules, yielding $g = O(b \log(n/b))$. These are questions of theoretical and also practical relevance, since for example there exist compressed indexes for highly repetitive collections that obtain different search performance depending on which compressibility measure their space is bounded by [27, Sect. 13.2].

Another relevant research avenue is to look for alternatives to Lempel-Ziv compression with a better approximation ratio. For example, a recent bidirectional scheme, *lcpcomp*, seems to always perform better than Lempel-Ziv in practice [9]. It would be interesting to research its approximation ratio with respect to the optimal bidirectional parsing.

Acknowledgements. We thank the reviewers for their insightful comments, which helped us improve the presentation significantly.

References

1. Belazzougui, D., Cunial, F.: Representing the suffix tree with the CDAWG. In: Proceedings of 28th Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 78, pp. 7:1–7:13 (2017)
2. Belazzougui, D., Cunial, F., Gagie, T., Prezza, N., Raffinot, M.: Composite repetition-aware data structures. In: Cicalese, F., Porat, E., Vaccaro, U. (eds.) CPM 2015. LNCS, vol. 9133, pp. 26–39. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19929-0_3

3. Bille, P., Gagie, T., Li Gørtz, I., Prezza, N.: A separation between run-length SLPs and LZ77. CoRR, abs/1711.07270 (2017)
4. Blumer, A., Blumer, J., Haussler, D., McConnell, R.M., Ehrenfeucht, A.: Complete inverted files for efficient text retrieval and analysis. J. ACM **34**(3), 578–595 (1987)
5. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation (1994)
6. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theory **51**(7), 2554–2576 (2005)
7. Cover, T., Thomas, J.: Elements of Information Theory, 2nd edn. Wiley, Hoboken (2006)
8. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rytter, W., Waleń, T.: Efficient algorithms for three variants of the LPF table. J. Discrete Algorithms **11**, 51–61 (2012)
9. Dinklage, P., Fischer, J., Köppl, D., Löbel, M., Sadakane, K.: Compression with the tudocomp framework. CoRR, abs/1702.07577 (2017)
10. Fici, G.: Factorizations of the Fibonacci infinite word. J. Integer Sequences, **18**(9), Article 3 (2015)
11. Fritz, M.H.-Y., Leinonen, R., Cochrane, G., Birney, E.: Efficient storage of high throughput DNA sequencing data using reference-based compression. Genome Res. **21**, 734–740 (2011)
12. Gagie, T.: Large alphabets and incompressibility. Inf. Process. Lett. **99**(6), 246–251 (2006)
13. Gallant, J.K.: String Compression Algorithms. Ph.D thesis. Princeton University (1982)
14. Gawrychowski, P.: Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. CoRR, abs/1104.4203 (2011)
15. Hucke, D., Lohrey, M., Reh, C.P.: The smallest grammar problem revisited. In: Inenaga, S., Sadakane, K., Sakai, T. (eds.) SPIRE 2016. LNCS, vol. 9954, pp. 35–49. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46049-9_4
16. I, T.: Longest common extensions with recompression. In: Proceedings of 28th Annual Symposium on Combinatorial Pattern Matching (CPM). LIPIcs, vol. 78, pp. 18:1–18:15 (2017)
17. Jez, A.: Approximation of grammar-based compression via recompression. Theor. Comput. Sci. **592**, 115–134 (2015)
18. Jez, A.: A really simple approximation of smallest grammar. Theor. Comput. Sci. **616**, 141–150 (2016)
19. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**(6), 918–936 (2006)
20. Kieffer, J.C., Yang, E.-H.: Grammar-based codes: a new class of universal lossless source codes. IEEE Trans. Inf. Theory **46**(3), 737–754 (2000)
21. Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Prob. Inf. Transm. **1**(1), 1–7 (1965)
22. Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theor. Comput. Sci. **483**, 115–133 (2013)
23. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Trans. Inf. Theory **22**(1), 75–81 (1976)
24. Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. Nord. J. Comput. **12**(1), 40–66 (2005)
25. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)

26. Mantaci, S., Restivo, A., Sciortino, M.: Burrows-Wheeler transform and Sturmian words. Inf. Process. Lett. **86**(5), 241–246 (2003)
27. Navarro, G.: Compact Data Structures - A Practical Approach. Cambridge University Press, Cambridge (2016)
28. Nishimoto, T., I, T., Inenaga, S., Bannai, H., Takeda, M.: Fully dynamic data structure for LCE queries in compressed space. In: Proceedings of 41st International Symposium on Mathematical Foundations of Computer Science (MFCS), pp. 72:1–72:15 (2016)
29. Prezza, N.: Compressed Computation for Text Indexing. Ph.D thesis. University of Udine (2016)
30. Rodeh, M., Pratt, V.R., Even, S.: Linear algorithm for data compression via string matching. J. ACM **28**(1), 16–24 (1981)
31. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003)
32. Sakamoto, H.: A fully linear-time approximation algorithm for grammar-based compression. J. Discrete Algorithms **3**(24), 416–430 (2005)
33. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 398–403 (1948)
34. Sthephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Chenxiang, Z., Efron, M.J., Iyer, R., Sinha, S., Robinson, G.E.: Big data: astronomical or genomical? PLoS Biol. **17**(7), e1002195 (2015)
35. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. J. ACM **29**(4), 928–951 (1982)

Kernelization for Maximum Happy Vertices Problem

Hang Gao[1(✉)] and Wenyu Gao[2]

[1] School of Transportation, Jilin University, Changchun, China
gaohang1998@163.com

[2] School of Information Science, Guangdong University of Finance and Economics, Guangzhou, China
gwy@gdufe.edu.cn

Abstract. The homophyly phenomenon is very common in social networks. The Maximum Happy Vertices (MHV) is a newly proposed problem related to homophyly phenomenon. Given a graph $G = (V, E)$ and a vertex coloring of G, we say that a vertex v is happy if v shares the same color with all its neighbors, and unhappy, otherwise, and that an edge e is happy, if its two endpoints have the same color, and unhappy, otherwise. Given a partial vertex coloring of G with k number of different colors, the k-MHV problem is to color all the remaining vertices such that the number of happy vertices is at least l. We study k-MHV from the parameterized algorithm perspective; we prove that k-MHV has an exponential kernel of $2^{kl+l} + kl + k + l$ on general graph. For planar graph, we get a much better polynomial kernel of $7(kl + l) + k - 10$.

Keywords: Maximum happy vertices · Happy coloring
Parameterized complexity · Kernelization · Planar graph

1 Introduction

Social networks attract more and more researchers. It is believed that homophyly is one of the most basic notions governing the structure of social networks. It is a common sense principle that people are more likely to connect with people they like.

Li and Peng [1] showed that many real networks satisfy exactly the homophyly law. Based on this, Zhang and Li [2] proposed a new problem that, given a network in which some vertices have their attributes unfixed, how to assign attributes to these vertices such that the resulting network reflects the homophyly law in the most degree? Suppose in a company there are many employees which constitutes a friendship network. Some employees have been assigned to work in some departments of the company, while the remaining employees are waiting to be assigned. An employee is happy, if she/he works in the same department with all of her/his friends; otherwise she/he is unhappy. Similarly, a friendship is happy if the two related friends work in the same department;

otherwise the friendship is unhappy. The goal is to achieve the greatest social benefits, that is, to maximize the number of happy vertices (similarly, happy edges) in the network.

The aforementioned problem can be formally expressed as follows. Consider a vertex-colored graph $G = (V, E)$, an edge is happy if its two endpoints have the same color (otherwise, the edge is unhappy). Similarly, a vertex is happy if it and all its neighbors have the same color (otherwise, the vertex is unhappy). Equivalently, a vertex is happy when all of its incident edges are happy. Let $S \subseteq V$, and let $c : S \to [k]$ be a partial vertex-coloring of G. A full coloring $c' : V \to [k]$ is an extended full coloring of c if $c(v) = c'(v)$ for all $v \in S$. In this paper, we consider the following coloring problems.

Definition 1. *k-Maximum Happy Vertices (k-MHV)*
Instance: A graph $G = (V, E)$, a partial vertex-coloring $c : S \to [k]$, an integer l.
Question: Is there an extended full coloring c' of c such that the number of the happy vertices is at least l?

Definition 2. *k-Maximum Happy Edges (k-MHE)*
Instance: A graph $G = (V, E)$, a partial vertex-coloring $c : S \to [k]$, an integer l.
Question: Is there an extended full coloring c' of c such that the number of the happy edges is at least l?

Though they are newly proposed algorithmic problems, both MHV and MHE problems are natural and fundamental algorithmic problems. The concept of homophyly reflected by it makes it important in social network application. Theoretical analysis of the problems helps to understand the nature of the problems, and hence is very welcome.

Zhang and Li [2] proved that for every $k \geq 3$, the problems k-MHE and k-MHV are NP-complete. However, when $k = 2$, they gave algorithms running in time $O(min\{n^{2/3}m; m^{3/2}\})$ and $O(mn^7 logn)$ for 2-MHE and 2-MHV, respectively. Towards this end, the authors used max-flow algorithms (2-MHE) and minimization of submodular functions (2-MHV). Moreover, the authors presented approximation algorithms with approximation ratios $1/2$ and $max\{1/k, \Omega(\Delta^{-3})\}$ for k-MHE and k-MHV, respectively, where Δ is the maximum degree of the graph. Later on, Zhang et al. [3] gave improved algorithms with approximation ratios 0.8535 and $1/(\Delta + 1)$ for k-MHE and k-MHV, respectively.

Aravind et al. [4] proved that both k-MHE and k-MHV are solvable in polynomial time for trees, and they proposed an interesting problem of studying the hardness of the k-MHV problem for planar graphs. Furthermore, Aravind et al. [5] proved an polynomial kernel for k-MHE problem from the parameterized algorithm perspective, they also proposed an open question that whether k-MHV admits a polynomial kernel.

Though the polynomial kernel for k-MHE problem proved by Aravind et al. [5] seems quite well, there is no kernelization algorithm for k-MHV problem currently, which inspires us to conduct a further study for k-MHV problem from the parameterized algorithm perspective.

2 Preliminaries

Throughout, we consider graphs that are finite, undirected and simple. For a graph $G = (V, E)$, let $V(G)$ denote its vertex set and $E(G)$ its set of edges. For $S \subseteq V$, the subgraph induced by S, denote $G[S]$, is the subgraph of G with vertex set S and edge set $\{(u, v)|u, v \in S, (u, v) \in E\}$. For each vertex $v \in V(G)$, let $N(v)$ be the set of vertices adjacent to v in G, i.e., $N(v) = \{u \in V|(u, v) \in E\}$, let $N_S(v)$ denote the set of vertices adjacent to v in S, where S is a subset of $V(G)$, i.e., $N_S(v) = \{x|x \in N(v) \text{ and } x \in S\}$.

The degree $deg(v)$ of v is the size of $N(v)$. A vertex of degree exactly (at most, at least) d is called a d-vertex ((d)-vertex, (d)-vertex). Let $\Delta(G)$ denote the maximum degree over all vertices in G.

If $u, v \in V$ are two vertices of graph G, the operation of contracting u and v is to combine u and v into a new vertex w, which becomes adjacent to all the former neighbors of u and of v.

A planar graph is a graph that can be embedded in the plane. For a given planar graph $G = (V, E)$, the following condition holds for $|V| \geq 3$, $|E| \leq 3|V| - 6$. The subdivision of an edge $e = (u, v)$ with endpoints u and v yields a graph containing one new vertex w, and with an edge set replacing e by two new edges, (w, u) and (w, v). For a planar graph, the subdivision of arbitrary edge will not change the planarity of the graph.

The theory of parameterized computation and complexity mainly considers decision problems (i.e., problems whose instances only require a yes/no answer). A parameterized problem Q is a decision problem (i.e., a language) that is a subset of $\Sigma^* \times N$, where Σ is a fixed alphabet and N is the set of all nonnegative integers. Thus, each element of Q is of the form (x, k), where the second component, i.e., the integer k, is the parameter. We say that an algorithm A solves the parameterized problem Q if on each input (x, k), the algorithm A can determine whether (x, k) is a yes-instance of Q (i.e., whether (x, k) is an element of Q). We call the algorithm A a parameterized algorithm if its computational complexity is measured in terms of both the input length $|x|$ and the parameter value k. The parameterized problem Q is fixed parameter tractable if it can be solved by a parameterized algorithm of running time bounded by $f(k)|x|^c$, where f is a recursive function and c is a constant independent of both k and $|x|$.

Kernelization is one of the most important techniques used in the development of efficient parameterized algorithm. Let Q be a parameterized problem and (x, k) be an instance of Q. An algorithm K is called a kernelization algorithm for Q if K satisfies the following conditions: (1) K transforms (x, k) into the reduced instance (x', k') in polynomial time; (2) (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q; and (3) $|x'| \leq g(k)$ and $k' \leq k$, where $g(k)$ is a computable function. Correspondingly, the problem Q is called kernelizable and the reduced instance (x', k') is called a kernel. In particular, Q is said to admit a polynomial kernel if $g(k)$ is a polynomial function on k. One of the most important theorems of parameterized computation is that a parameterized problem is fixed parameter tractable if and only if it is kernelizable [6].

The reduced kernel is not only helpful for parameterized algorithm, but also helpful for approximation algorithm. For more on parameterized complexity, we refer the interested reader to [7,8].

3 Kernelization for k-MHV Problem

According to the description of k-MHV, the difficulty is that since the original graph is a partial-colored graph, it is very hard to determine whether an colored or uncolored vertex is a happy vertex of the final optimal solution. If the graph is dense enough (for instance, complete graph), only two pre-colored vertices with different colors will make all the rest vertices be unhappy vertices. On the contrary, given a sparse graph, it is easier to process.

From the perspective of parameterized algorithm, kernelization is to reduce the original graph G to a "small" kernel. In the following, we prove k-MHV has a exponential kernel of $2^{kl+l} + kl + k + l$, thus k-MHV is fixed parameter tractable. Furthermore, in planar graph, k-MHV has a much better kernel of $7(kl + l) + k - 10$. Our strategy to obtain the kernels consists of three parts: firstly, we partition the vertices of the partial-colored graph into six disjoint sets, secondly, we introduce four reduction rules to reduce the graph; thirdly, we introduce a strategy to analyze the size of each disjoint set of vertices.

3.1 Partition of Vertices

Given a partial-colored graph $G = (V, E)$, we can partition the vertices of G into six disjoint sets.

(1) Set of colored happy vertices, denoted by V_1, i.e., $v \in V_1$ if v is colored and all the vertices in $N(v)$ have been colored with the same color as that of v.
(2) Set of colored unhappy vertices, denoted by V_2, i.e., $v \in V_2$ if v is colored and destined to be unhappy. That is to say, there is at least one vertex $u \in N(v)$, which has been colored with the color different from that of v.
(3) Set of colored potential happy vertices, denoted by V_3, i.e., $v \in V_3$ if v is colored and v has the same color as all of its colored neighbors, but v also has some uncolored neighbors.
(4) Set of uncolored unhappy vertices, denoted by V_4, i.e., $v \in V_4$ if v is uncolored and v has at least two colored neighbors with different colors.
(5) Set of uncolored potential happy vertices, denoted by V_5, i.e., $v \in V_5$ if v is uncolored and all the colored neighbors of v have the same color, but v also has some uncolored neighbors.
(6) Set of uncolored free vertices, denoted by V_6, i.e., $v \in V_6$ if v is uncolored and v has no colored neighbors.

The partition of the vertices is shown in Fig. 1. Black vertices are colored vertices; the numbers beside those colored vertices represent their color numbers. White vertices are uncolored vertices.

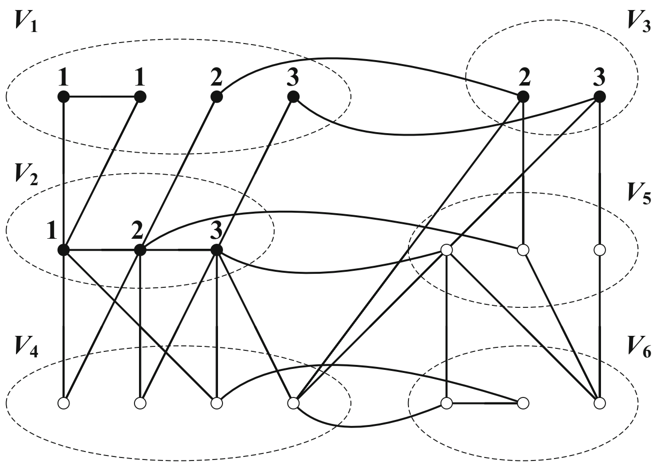


Fig. 1. Partition of vertices of G. V_1 is the set of colored happy vertices, V_2 is the set of colored unhappy vertices, V_3 is the set of colored potential happy vertices, V_4 is the set of uncolored unhappy vertices, V_5 is the set of uncolored potential happy vertices, V_6 is the set of uncolored free vertices.

3.2 Reduction Rules

Now we present the following four reduction rules.

Rule 1: Remove all the happy edges, because the happy edges will never further affect the coloring of the remaining vertices. By removing the happy edges, all the colored happy vertices (vertices in V_1) in graph G become isolated vertices. Then remove all the isolated vertices and decrease l by the number of removed vertices. By doing so, the set of V_1 no longer exists in the reduced graph.

Rule 2: Considering the colored unhappy vertices in V_2, contract all the vertices with the same color into a single vertex and remove all but one of the parallel edges.

Proof. According to the partition of the vertices, vertices in V_2 are unhappy vertices; they will never further contribute to the value of the optimal solution. How many colored vertices are adjacent to an uncolored vertex is not serious, but what colored vertices are adjacent to an uncolored vertex is the key problem. Moreover, after applying of Rule 1, all the vertices with the same color in V_2 forms an independent set. Therefore, we can contract all the vertices with the same color into a single vertex and remove the parallel edges safely.

Rule 3: If there are some vertices in V_4 only adjacent to vertices in V_2, remove them from G.

Proof. The vertices in V_4 only adjacent to vertices in V_2 will never affect the colored vertices in V_3, the uncolored vertices in V_5, and the free vertices in V_6. So they can be removed in advance.

Rule 4: For the rest of uncolored unhappy vertices in V_4, they are all adjacent to vertices in V_3 or V_5, or vertices in V_6. That is to say, they are adjacent to at least one vertex in $V_3 \cup V_5 \cup V_6$. If there are two vertices $u, v \in V_4$, and $N_{V_3 \cup V_5 \cup V_6}(u) \subseteq N_{V_3 \cup V_5 \cup V_6}(v)$, then remove vertex u.

Proof. Vertices in V_4 are destined to be unhappy vertices, so in the optimal solution, they can be colored with the same color as one of its neighbors in $V_3 \cup V_5 \cup V_6$ safely. Suppose in the optimal solution, vertex u is colored with color c_1, and vertex v is colored with color c_2, thus all the neighbors of u are unhappy vertices. So it is safe to change the color of u from c_1 to c_2, which will not decrease the number of happy vertices. Thus if there are two vertices u and v in V_4 such that $N_{V_3 \cup V_5 \cup V_6}(u) \subseteq N_{V_3 \cup V_5 \cup V_6}(v)$, the best choice is to color them with the same color. Because u and v will be colored with the same color in the end, and $N_{V_3 \cup V_5 \cup V_6}(u) \subseteq N_{V_3 \cup V_5 \cup V_6}(v)$, then u can be removed in advance.

Therefore, after applying of Rule 4, for each pair of vertices $u, v \in V_4$, the following two forms are satisfied.

$$\begin{aligned} N_{V_3 \cup V_5 \cup V_6}(u) \nsubseteq N_{V_3 \cup V_5 \cup V_6}(v) \\ N_{V_3 \cup V_5 \cup V_6}(u) \nsupseteq N_{V_3 \cup V_5 \cup V_6}(v) \end{aligned} \tag{1}$$

The reduced graph G' by applying Rule 1 to Rule 4 is shown in Fig. 2. It is worth noting that the happy edges and the colored happy vertices have been removed from the graph.

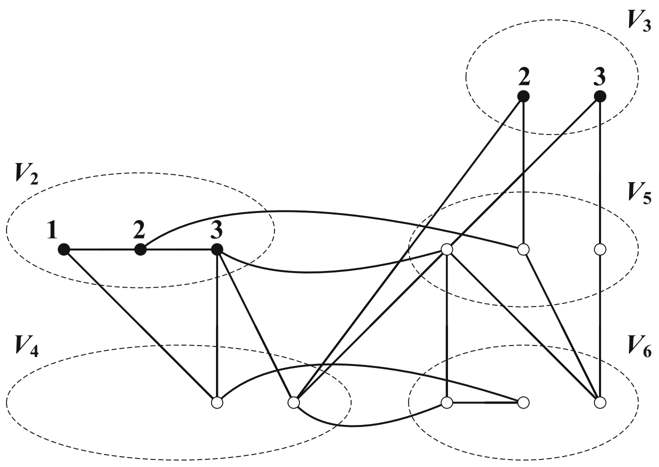


Fig. 2. The reduced graph G'. Black vertices are colored vertices; the numbers beside these colored vertices are the colors of them. White vertices are uncolored vertices.

3.3 Analyzing Size of Vertex Set

After applying the aforementioned four reduction rules, we are ready to conclude the kernel of k-MHV.

Theorem 3. *The problem k-MHV admits a kernel of* $2^{(k+l)} + kl + k + l$ *vertices. Thus, k-MHV is fixed parameter tractable.*

Proof. Let (G', k, l) be a reduced instance of k-MHV. We claim that if G' has more than $2^{(k+l)} + kl + k + l$ vertices, then we have YES-instance. The proof follows by the claims below.

Claim 1.1. The number of vertices in V_1 is 0, i.e.,

$$|V_1| = 0 \tag{2}$$

It is obvious according to Rule 1.

Claim 1.2. The number of vertices in V_2 is at most k, i.e.,

$$|V_2| \leq k \tag{3}$$

Proof. By applying Rule 2, all the vertices with the same color are contracted into a single vertex, so there are at most k vertices with different colors in V_2.

Claim 1.3. The number of free vertices in V_6 is at most $l - 1$, i.e.,

$$|V_6| < l \tag{4}$$

Proof. For the uncolored free vertices in V_6, if there are at least l uncolored free vertices, we can conclude that there is a solution that contains at least l happy vertices. Because in that case, we can color all the uncolored vertices of graph G' with the same color, which makes all the uncolored free vertices in V_6 become happy vertices. So $|V_6| < l$.

Claim 1.4. The number of vertices in $V_3 \cup V_5$ is at most $kl - 1$, i.e.,

$$|V_3 \cup V_5| < kl \tag{5}$$

Proof. For arbitrary uncolored potential happy vertex $u, u \in V_5$, if the color number of its colored neighbors is c_1, then we call c_1 as the potential color number of vertex u. If there are at least kl potential happy vertices (colored or uncolored) in $V_3 \cup V_5$, we can choose at least $kl/k = l$ vertices whose color number or potential color number are the same, because there are at most k different color numbers. Then we can make these l vertices become happy vertices. This is feasible because we can color all of them and their neighbors with their original color number or potential color number. Thus, $|V_3 \cup V_5| < kl$.

Claim 1.5. The number of vertices in V_4 is at most 2^{kl+l}, i.e.,

$$|V_4| < 2^{kl+l} \tag{6}$$

Proof. By applying Rules 3 and 4, each vertex in V_4 is adjacent to at least one vertex in $V_3 \cup V_5 \cup V_6$, and for each pair of vertices $u, v \in V_4$, $N_{V_3 \cup V_5 \cup V_6}(u) \nsubseteq N_{V_3 \cup V_5 \cup V_6}(v)$, and $N_{V_3 \cup V_5 \cup V_6}(u) \nsupseteq N_{V_3 \cup V_5 \cup V_6}(v)$. According to Claims 1.3 and 1.4, $|V_3 \cup V_5 \cup V_6| < kl + l$. Because the set of $V_3 \cup V_5 \cup V_6$ has at most $\binom{kl+l}{(kl+l)/2}$ number of subsets that do not contain each other. Thus, $|V_4| < \binom{kl+l}{(kl+l)/2} < 2^{kl+l}$.

Therefore, the number of vertices after applying reduction rules is,

$$|V_1 \cup V_2 \cup V_3 \cup V_4 \cup V_5 \cup V_6| < 0 + k + kl + 2^{kl+l} + l = 2^{kl+l} + kl + k + l \quad (7)$$

This is an exponential kernel of k-MHV problem. Furthermore, all of the reduction rules can be implemented to run in polynomial time. Thus, the claimed kernel follows. □

According to Rule 1 to Rule 4 and Theorem 3, we can easily design a kernelization algorithm for k-MHV problem, the kernelization algorithm just need to include the four reduction rules.

3.4 A k-approximation Algorithm

It is worth noting that the strategy used to bound $|V_3 \cup V_5|$ in proof of Theorem 3 (Claim 1.4) can be extended to an approximation algorithm. The main idea is that we can choose at least $|V_3 \cup V_5|/k$ vertices whose color number or potential color number are the same, and make these vertices become happy vertices by coloring all of them and their neighbors with their original color number or potential color number.

So we can change at least $|V_3 \cup V_5|/k$ vertices into happy vertices every time. If $V_3 \cup V_5$ is null, we only need to color all the vertices in V_6 and all their neighbors in V_4 with the same color, which lead to the end of the approximation algorithm.

Given a partial-colored graph G, the vertices in V_2 and V_4 will never become happy vertices, the vertices in V_1 will not affect the rest coloring procedure. So in the best case, we can assume that the vertices in V_3, V_5, and V_6 will all become happy vertices in the end. But by our approximation algorithm, we can make at least $|V_3 \cup V_5|/k$ vertices into happy vertices every time, till the end. Thus this is a k-approximation algorithm.

4 Maximum Happy Vertices on Planar Graph

It is obvious that the graph is more dense, there are fewer happy vertices. Thus, it is natural to ask what the parameterized complexity of k-MHV on planar graph is. For a planar graph $G = (V, E)$, $|E| \leq 3|V| - 6$.

Zhang and Li [2] proved the NP-hardness of k-MHE by reducing multiway cut problem to k-MHE, and they also proved the NP-hardness of k-MHV by reducing k-MHE to k-MHV. Dahlhaus et al. [9] proved that the multiway cut problem is NP-hard even on planar graphs. Thus, it is obvious that k-MHV on planar graphs is also NP-hard. To bound the kernel of k-MHV on planar graph, we use a method proposed by Wang et al. [10] to count the vertices on planar graph.

The proof of Theorem 3 shows that the cardinalities of V_3, V_5, and V_6 are bounded to polynomial functions of k or l, but the cardinality of V_4 is related to an exponential function of k and l. Thus, our main idea is to bound the cardinality of V_4 by taking advantage of the properties of planar graph.

Let's consider the subgraph containing Vertex set V_4 and $V_3 \cup V_5 \cup V_6$, and the edges between them. It is a bipartite graph, we denote it by $B = (V_4 \cup V_{3-5-6}, E)$, where $V_{3-5-6} = V_3 \cup V_5 \cup V_6$. At the beginning, if the given graph G is a planar graph, during the reduction of graph G, applying reduction Rule 1 will not change the planarity of graph G, applying reduction Rule 2 may change the planarity of graph G, but it will not affect the planarity of the induced subgraph $G[V_4 \cup V_3 \cup V_5 \cup V_6]$. The aforementioned bipartite graph $B = (V_4 \cup V_{3-5-6}, E)$ is a subgraph of $G[V_4 \cup V_3 \cup V_5 \cup V_6]$, so $B = (V_4 \cup V_{3-5-6}, E)$ is a bipartite planar graph. Clearly, applying of Rules 3 and 4 will remove some vertices in V_4, which will not change the planarity of $B = (V_4 \cup V_{3-5-6}, E)$.

Therefore, given a planar graph G, after applying reduction rules, the reduced graph G' may be a non-planar graph, but the subgraph $B = (V_4 \cup V_{3-5-6}, E)$ of G' is a bipartite planar graph.

Based on this condition, we arrive at the kernel size of k-MHV on planar graph.

Theorem 4. *The problem k-MHV on planar graph admits a linear kernel of* $7(kl + l) + k - 10$ *vertices.*

Proof. In Theorem 3, we prove that $|V_2| \le k$, $|V_3 \cup V_5| < kl$, $|V_6| < l$ after using four reduction rules. Now let us count V_4 on the condition of the subgraph $B = (V_4 \cup V_{3-5-6}, E)$ is a planar graph.

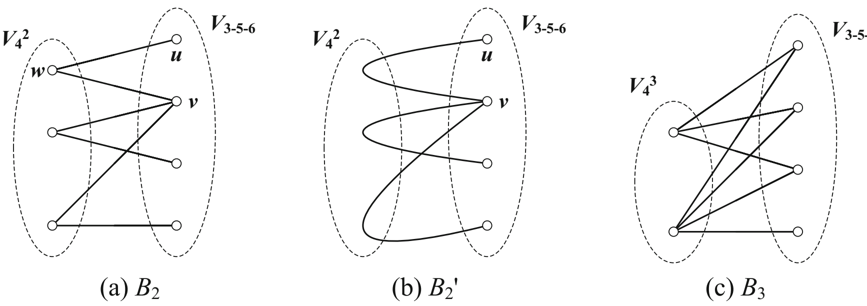
Firstly, we partition V_4 into three disjoint sets, V_4^1 denotes the set of vertices which have only one neighbor in V_{3-5-6}, V_4^2 denotes the set of vertices which have exact two neighbors in V_{3-5-6}, and V_4^3 denotes the set of vertices which have at least three neighbors in V_{3-5-6}.

Secondly, let's bound the number of vertices in V_4^1. Because of each vertex in V_4 is adjacent to at least one vertex of V_{3-5-6}, and for each pair of vertices $u, v \in V_4$, $N_{V_3 \cup V_5 \cup V_6}(u) \nsubseteq N_{V_3 \cup V_5 \cup V_6}(v)$, and $N_{V_3 \cup V_5 \cup V_6}(u) \nsupseteq N_{V_3 \cup V_5 \cup V_6}(v)$. Thus, we have

$$|V_4^1| \le |V_{3-5-6}| = kl + l \tag{8}$$

Thirdly, let's bound the number of vertices in V_4^2. Each vertex in V_4^2 is adjacent to different pair of vertices in V_{3-5-6}, and $B = (V_4 \cup V_{3-5-6}, E)$ is a bipartite planar graph. Let's consider the subgraph of B formed by V_4^2, V_{3-5-6}, and the edges between V_4^2 and V_{3-5-6}. We denote it by B_2, a example is shown in Fig. 3(a). It is clear that this subgraph B_2 is also a bipartite planar graph, and for each pair of vertices $u, v \in V_{3-5-6}$, there is at most one vertex $w \in V_4^2$ adjacent to both u and v (because $N_{V_3 \cup V_5 \cup V_6}(u) \nsubseteq N_{V_3 \cup V_5 \cup V_6}(v)$, and $N_{V_3 \cup V_5 \cup V_6}(u) \nsupseteq N_{V_3 \cup V_5 \cup V_6}(v)$). Assuming we replace every vertex $w \in V_4^2$ and its two incident edges (w, u) and (w, v) by a single edge (u, v) (this is a reverse operation of subdivision), let B_2' denote the new graph by replacing operation, a example is shown in Fig. 3(b). Clearly, B_2' is a planar graph whose vertex set is V_{3-5-6}. Moreover, there is a one-to-one correspondence between the edge set of B_2' and the replaced vertices in V_4^2. Since every planar graph with $|V|$ vertices and $|E|$ edges satisfies $|E| \leq 3|V| - 6$, therefore,

$$|V_4^2| \leq 3|V_{2-4-5}| - 6 = 3(kl + l) - 6 \tag{9}$$



V_4^2
$V_{3\text{-}5\text{-}6}$
w
u
v
(a) B_2
V_4^2
$V_{3\text{-}5\text{-}6}$
u
v
(b) B_2'
V_4^3
$V_{3\text{-}5\text{-}6}$
(c) B_3

Fig. 3. Examples of bipartite planar graphs, (a) B_2 is a bipartite planar graph formed by V_4^2, V_{3-5-6}, and the edges between V_4^2 and V_{3-5-6}, (b) B_2' is transformed from B_2 by replacing every vertex $w, w \in V_4^2$ and its two incident edges (w, u) and (w, v) by a single edge (u, v), (c) B_3 is a bipartite planar graph formed by V_4^3, V_{3-5-6}, and the edges between V_4^3 and V_{3-5-6}.

Fourthly, let's bound the number of vertices in V_4^3. Each vertex in V_4^3 is adjacent to at least three vertices in V_{3-5-6}, and $B = (V_4 \cup V_{3-5-6}, E)$ is a bipartite planar graph. Considering the subgraph of B formed by V_4^3, V_{3-5-6}, and the edges between V_4^3 and V_{3-5-6}, denoted by B_3, shown in Fig. 3(c). Clearly, this subgraph is also a bipartite planar graph, which is a triangle-free planar graph. Since triangle-free planar graph with $|V|$ vertices and $|E|$ edges satisfies $|E| \leq 2|V| - 4$, let's consider the edges between V_4^3 and V_{3-5-6}, we have

$$\begin{aligned} 3|V_4^3| &\leq |E| \leq 2(|V_4^3| + |V_{3-5-6}|) - 4 \\ \Rightarrow |V_4^3| &\leq 2|V_{3-5-6}| - 4 \\ \Rightarrow |V_4^3| &\leq 2(kl + l) - 4 \end{aligned} \tag{10}$$

Therefore,

$$\begin{aligned} |V_4| &= |V_4^1 \cup V_4^2 \cup V_4^3| \\ &= |V_4^1| + |V_4^2| + |V_4^3| \\ &\leq 6(kl + l) - 10 \end{aligned} \tag{11}$$

After reduction, the number of vertices is,

$$|V_1 \cup V_2 \cup V_3 \cup V_4 \cup V_5 \cup V_6| < 0 + k + kl + l + 6(kl + l) - 10 = 7(kl + l) + k - 10 \tag{12}$$

So k-MHV on planar graph admits a kernel of $7(kl + l) + k - 10$. $\square$

5 Conclusions

The k-MHV problem is a natural graph coloring problem arising in the homophyly phenomenon of networks. By vertex partition, we prove an exponential kernel of $2^{kl+l} + kl + k + l$ for this problem. For planar graph, we can achieve a much better polynomial kernel of $7(kl + l) + k - 10$.

In general, many NP-hard problems are less difficult in some special graphs, and studying of NP-hard problems in special graphs can be very helpful for solving these problems in general graphs. Considering MHV is a newly proposed problem, it is necessary to investigate the complexity of MHV in more special classes of graphs.

Acknowledgments. We would like to thank the anonymous reviewers for their detailed reviews and suggestions.

References

1. Li, A., Peng, P.: The small-community phenomenon in networks. Math. Struct. Comput. Sci. **22**(3), 373–407 (2012)
2. Zhang, P., Li, A.: Algorithmic aspects of homophyly of networks. Theor. Comput. Sci. **593**(C), 117–131 (2015)
3. Zhang, P., Jiang, T., Li, A.: Improved approximation algorithms for the maximum happy vertices and edges problems. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 159–170. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_13
4. Aravind, N.R., Kalyanasundaram, S., Kare, A.S.: Linear time algorithms for happy vertex coloring problems for trees. In: Mäkinen, V., Puglisi, S.J., Salmela, L. (eds.) IWOCA 2016. LNCS, vol. 9843, pp. 281–292. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44543-4_22
5. Aravind, N.R., Kalyanasundaram, S., Kare, A.S., Lauri, J.: Algorithms and hardness results for happy coloring problems. CoRR abs/1705.08282 (2017)
6. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2002)
7. Chen, J.E.: Parameterized computation and complexity: a new approach dealing with np-hardness. J. Comput. Sci. Technol. **20**(1), 18–37 (2005)
8. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer International Publishing, Heidelberg (2015)
9. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM J. Comput. **23**(4), 864–894 (1994)
10. Wang, J., Yang, Y., Guo, J., Chen, J.: Planar graph vertex partition for linear problem kernels. J. Comput. Syst. Sci. **79**(5), 609–621 (2013)

When is Red-Blue Nonblocker Fixed-Parameter Tractable?

Serge Gaspers[1,2(✉)], Joachim Gudmundsson[3], Michael Horton[3], and Stefan Rümmele[1,3]

[1] UNSW, Sydney, Australia
{sergeg,stefanr}@cse.unsw.edu.au
[2] Data61, CSIRO, Canberra, Australia
[3] University of Sydney, Sydney, Australia
joachim.gudmundsson@gmail.com, michael.horton@sydney.edu.au

Abstract. In the Red-Blue Nonblocker problem, the input is a bipartite graph $G = (R \uplus B, E)$ and an integer k, and the question is whether one can select at least k vertices from R so that every vertex in B has a neighbor in R that was not selected. While the problem is W[1]-complete for parameter k, a related problem, Nonblocker, is FPT for parameter k. In the Nonblocker problem, we are given a graph H and an integer k, and the question is whether one can select at least k vertices so that every selected vertex has a neighbor that was not selected. There is also a simple reduction from Nonblocker to Red-Blue Nonblocker, creating two copies of the vertex set and adding an edge between two vertices in different copies if they correspond to the same vertex or to adjacent vertices. We give FPT algorithms for Red-Blue Nonblocker instances that are the result of this transformation – we call these instances *symmetric*. This is not achieved by playing back the entire transformation, since this problem is NP-complete, but by a kernelization argument that is inspired by playing back the transformation only for certain well-structured parts of the instance. We also give an FPT algorithm for almost symmetric instances, where we assume the symmetry relation is part of the input.

Next, we augment the parameter by $\ell = |B| / |R|$. Somewhat surprisingly, Red-Blue Nonblocker is W[1]-hard for the parameter $k+\ell$, but becomes FPT if no vertex in B has degree 1. The FPT algorithm relies on a structural argument where we show that when $|R|$ is large with respect to k and ℓ, we can greedily compute a red-blue nonblocker of size at least k. The same results also hold if we augment the parameter by d_R instead of ℓ, where d_R is the average degree of the vertices in R.

1 Introduction

In this paper, we study the Red-Blue Nonblocker problem.

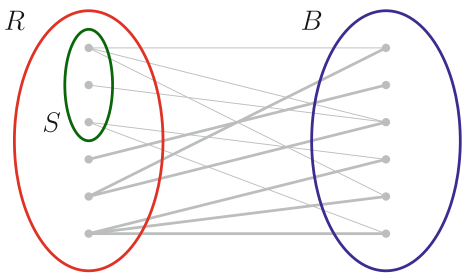


Fig. 1. Example of a red-blue nonblocker S. Each vertex in B has an escape to a vertex in $R \setminus S$, denoted by the thick edges. (Color figure online)

RED-BLUE NONBLOCKER	Parameter: k
Input: Bipartite graph $G = (R \uplus B, E)$, integer k	
Question: Is there a set $S \subseteq R$ with $\|S\| \geq k$ such that for each vertex $v \in B$, $N(v) \not\subseteq S$?	

We call such a set S a *red-blue nonblocker* or simply a *nonblocker*. See Fig. 1 for a simple example of a red-blue nonblocker. We also refer to vertices in R as *red* vertices and to vertices in B as *blue* vertices. For a red-blue nonblocker S, we have that each blue vertex v has a red neighbor $u \notin S$, and we refer to such a neighbor as an *escape* for v.

The problem can be viewed as an alternative parameterization of the SET COVER problem and of the RED-BLUE DOMINATING SET problem: a set $S \subseteq R$ is a red-blue nonblocker if and only if $R \setminus S$ is a red-blue dominating set; if we view the set B as elements and the set R as sets containing their neighboring vertices as elements, then $S \subseteq R$ is a red-blue nonblocker if and only if $R \setminus S$ is a set cover.

The RED-BLUE NONBLOCKER problem is W[1]-complete [6], but it is closely related to the following problem, which is FPT [3].

NONBLOCKER	Parameter: k
Input: Graph $H = (V, E)$, integer k	
Question: Is there a set $S \subseteq V$ with $\|S\| \geq k$ such that for each vertex $v \in V$, $N[v] \not\subseteq S$?	

Here, $N[v]$ denotes the closed neighborhood of v.

There is a simple reduction transforming instances of NONBLOCKER into equivalent instances of RED-BLUE NONBLOCKER (see, e.g., [9] where the reduction is formulated in terms of dominating sets and set covers): for each vertex $v \in V$, create a vertex $v_r \in R$ and a vertex $v_b \in B$; two vertices v_r, u_b are adjacent in G if $u \in N_H[v]$; the value of k is the same in both instances. We call graphs resulting from this transformation symmetric graphs.

Definition 1. *A bipartite graph $G = (R \uplus B, E)$ is* symmetric *if there are permutations $(r_1, \ldots, r_{n/2})$ and $(b_1, \ldots, b_{n/2})$ of R and B, respectively, such that $r_i b_i \in E$ for each $i \in \{1, \ldots, n/2\}$ and $r_i b_j \in E$ if and only if $r_j b_i \in E$. A* symmetry relation *of G maps r_i to b_i and b_i to r_i, for all $i \in \{1, \ldots, n/2\}$, in such permutations.*

We note that this definition is slightly different from the one in [2], where it is not necessary that $r_i b_i \in E$.

When we ask the question for which kinds of graphs Red-Blue Nonblocker could be FPT, symmetric graphs immediately come to mind. In fact, if the input graph G is symmetric, and we have as additional input a symmetry relation for G, we can rewind the above transformation and reduce the problem to Nonblocker to get an FPT algorithm. But what if we do not have this additional input? We could try and compute a symmetry relation for G. However, this turns out to be a difficult problem. In fact, even deciding whether a bipartite graph is symmetric is NP-complete (Theorem 1).

Nevertheless, a closer inspection of the FPT algorithm for Nonblocker from [3] reveals that we only need to be able to detect symmetries for vertices of small degree, and we can adapt their algorithm to show that Red-Blue Nonblocker is FPT if the input graph is symmetric (Theorem 2). We can go even further and consider almost symmetric graphs. These are bipartite graphs that can be turned into symmetric graphs by deleting a small number of vertices. Parameterizing by k plus the number of deletions, we obtain an FPT algorithm for this variant if we are also given the symmetry relation of the resulting symmetric graph as additional input (Theorem 3). The problem is akin to an annotated variant of Nonblocker. If, in the above transformation, we omit the red copy of a vertex, this means that we have already decided that the corresponding vertex is in the nonblocker. If, instead, we omit the blue copy, this means that we omit the requirement that the corresponding vertex needs a neighbor that is not in the nonblocker. We also observe that [3] already handled the annotation where a vertex is not allowed to be added to the nonblocker, without any significant overhead in running time (all such annotations can be removed by increasing the parameter by 3).

Next, we consider an additional parameter, the ratio $\ell = |B|/|R|$. The Red-Blue Nonblocker problem remains W[1]-hard for parameter $k + \ell$. But we also present an FPT algorithm for the case where no vertex in B has degree 1. For this, we define an auxiliary graph that captures only some of the neighborhood relations of the original graph, but enough of them so that when $|R|$ is large with respect to k and ℓ, we are sure to have a Yes-instance. When $|R|$ is small with respect to k and ℓ, we use a brute-force approach. This results in an FPT algorithm for Red-Blue Nonblocker on instances with no blue vertices of degree 1 for the parameter $k+\ell$. The running time of this algorithm is $\binom{(2\ell+1)k}{k} n^{O(1)}$ (see Theorem 4). As a by-product we also obtain the same results for the parameter $k + d_R$, where d_R denotes the average degree of the vertices in R (Corollary 2).

1.1 Related Work

For each constant $s \geq 2$, the variant of Red-Blue Nonblocker where the degree of each vertex in B is upper bounded by s remains W[1]-complete [6, Theorem 21.2.5], but when all vertices have degree at most s, the problem is FPT [6, Exercise 21.3.1].[1] The problem played a major role in establishing the initial theory around the complexity class W[1] [4]. Kanj and Xia [12] showed that Red-Blue Nonblocker is FPT when G has genus $|R|^{o(1)}$ but remains W[1]-complete on graphs of genus $|R|^{\Omega(1)}$. For parameter $|R|$, the problem is trivially FPT; for parameter $|B|$, the problem has a dynamic programming algorithm with running time $2^{|B|}n^{O(1)}$ [8]; for the parameter $|R| - k$, the problem is W[2]-complete [6]; and for the parameter treewidth the problem can be solved in time $3^{\ell}n^{O(1)}$ if a tree decomposition of width ℓ is provided as part of the input [1].

Ore [15] proved that for a minimal dominating set $S \subseteq V$ of a graph $G = (V, E)$ with no isolated vertices, the set $V \setminus S$ is a dominating set. Given that isolated vertices can never be added to a nonblocker, and Ore's result implies that a graph $G = (V, E)$ with minimum degree at least one has a dominating set of size at most $|V|/2$, we have that Nonblocker has a kernel with $2k$ vertices. This observation was made by [3] who improved the kernel size to $5k/3+3$ vertices and gave an FPT algorithm with running time $2.5154^k n^{O(1)}$. The kernel is based on a result by McCuaig and Shepherd [14] that every connected graph with $n \geq 8$ vertices has a dominating set of size at most $2n/5$. Their kernel combined with the currently fastest algorithm for Nonblocker (or Dominating Set) measured in terms of the number of vertices by Iwata [11] gives an algorithm for Nonblocker with running time $1.8982^k k^{O(1)} + n^{O(1)}$.

2 Symmetric Bipartite Graphs

We first show that we cannot simply reverse the transformation from Nonblocker to Red-Blue Nonblocker without being provided the symmetry relation, since even checking whether a bipartite graph is symmetric is NP-complete. The proof is by a reduction from the NP-complete problem List Restricted Graph Isomorphism [13].

Theorem 1. *Deciding whether a bipartite graph is symmetric is NP-complete.*

Proof. Membership in NP follows by a guess-and-check algorithm.

For hardness, we reduce from the NP-complete problem List Restricted Graph Isomorphism [13]. An instance of this problem consists of two graphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ and each vertex $v \in V_1$ has a list $L(v) \subseteq V_2$. The question is whether there exists an isomorphism mapping H_1 to H_2 such that $v \in V_1$ is mapped to one its list entries $L(v)$. We transform an instance of

[1] Actually, the variant where all vertices have degree at most s was thought to be W[1]-complete for a long time [5], and Downey and Fellows [6] report that Alexander Vardy spotted a flaw in their initial proof.

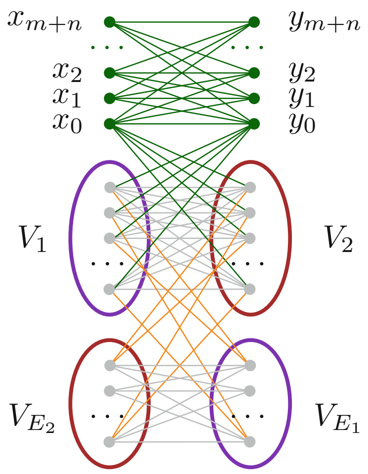


Fig. 2. Example of reduction of instance of Graph Isomorphism to instance of symmetry detection. The induced graph is bipartite and symmetric if H_1 and H_2 are isomorphic, as is the case here.

this problem to an instance $G = (R \uplus B, E)$ of our symmetry detection problem as follows. Let V_{E_1} and V_{E_2} be sets of new vertices corresponding to edges in the original graphs, that is $V_{E_1} = \{v_e \mid e \in E_1\}$ and $V_{E_2} = \{v_e \mid e \in E_2\}$. Let $X = \{x_0, \ldots, x_{m+n}\}$ and $Y = \{y_0, \ldots, y_{m+n}\}$ be two sets of $m + n + 1$ new vertices, where $n = |V_1| = |V_2|$ and $m = |E_1| = |E_2|$. Each side of G consists of the vertices of one of the original graphs together with vertices corresponding to edges of the other original graph and one of the two sets of fresh vertices, that is $R = V_1 \cup V_{E_2} \cup X$ and $B = V_2 \cup V_{E_1} \cup Y$. The edge set E contains the following edges:

1. $\{v_1, v_2\} \in E$ for all $v_1 \in V_1$ and $v_2 \in L(v_1)$,
2. $\{v_{e_1}, v_{e_2}\} \in E$ for all $e_1 \in E_1$ and $e_2 \in E_2$,
3. $\{u, v_e\} \in E$ and $\{v, v_e\} \in E$ for all $e = \{u, v\} \in E_1 \cup E_2$,
4. $\{x, y\} \in E$ for all $x \in X$ and $y \in Y$,
5. $\{x_0, v\} \in E$ for all $v \in V_2$, and
6. $\{y_0, v\} \in E$ for all $v \in V_1$.

See Fig. 2 for an example of this construction. This reduction can be done in polynomial time.

For correctness, assume that G is symmetric, that is, there exists a symmetry relation ϕ of G. We will show that there is an isomorphism from H_1 to H_2 satisfying the constraints expressed in the lists $L(v)$. A symmetry relation can map vertices only if they have the same degree. By construction that means that vertices in X can only be mapped to vertices in Y. Specifically, x_0 has to be mapped to y_0. Since x_0 is connected to V_2 and y_0 is connected to V_1, we know that vertices in V_1 can only be mapped to vertices in V_2 and vice versa. Hence, the remaining vertices in V_{E_1} are mapped to vertices in V_{E_2} and vice versa. We can define a bijection $f : V_1 \to V_2$ by setting $f(v) = u$ for $v \in V_1$ with $\{v, u\} \in \phi$. Hence, f satisfies the constraints expressed by lists $L(v)$. We will show that f is edge-preserving. Let $u, v \in V_1$ with $e = \{u, v\} \in E_1$. Let $f(u), f(v) \in V_2$ be the corresponding vertices in V_2 and let $e' \in E_2$ be the edge such that $\{v_e, v_{e'}\} \in \phi$. Since ϕ is a symmetry relation, we know that $v_{e'}$ is connected to $f(u)$ and $f(v)$. Hence, by construction, e' is the edge between $f(u)$ and $f(v)$ in H_2. Therefore, f is a witness for our List Restricted Graph Isomorphism problem.

For the other direction, assume that H_1 and H_2 are isomorphic. Let $f : V_1 \to V_2$ be the corresponding edge-preserving bijection that satisfies the lists $L(v)$. We can construct the symmetry relation ϕ of G as follows:

1. $\{x_i, y_i\} \in \phi$ for $0 \leq i \leq n + m$.
2. $\{v, f(v)\} \in \phi$ for all $v \in V_1$.
3. $\{e_1, e_2\} \in \phi$ where $e_1 = \{u, v\} \in E_1$ and $e_2 = \{f(u), f(v)\}$.

Hence, G is symmetric and the reduction is correct. □

In our FPT results we will make use of preprocessing using some of the reduction rules below. While not all of them are needed to achieve our time bounds, we nevertheless list them here. In practice, one would want to simplify the instance as much as possible before running the respective FPT algorithms. In what follows, we denote the degree of a vertex v by $d(v)$. Figure 3 illustrates each of the reduction rules.

Reduction rule 1 (degree 0 blue vertex). If there exists $v \in B$ with $d(v) = 0$, then answer No.

Reduction rule 2 (degree 0 red vertex). If there exists $v \in R$ with $d(v) = 0$, then return $(G \setminus \{v\}, k - 1)$.

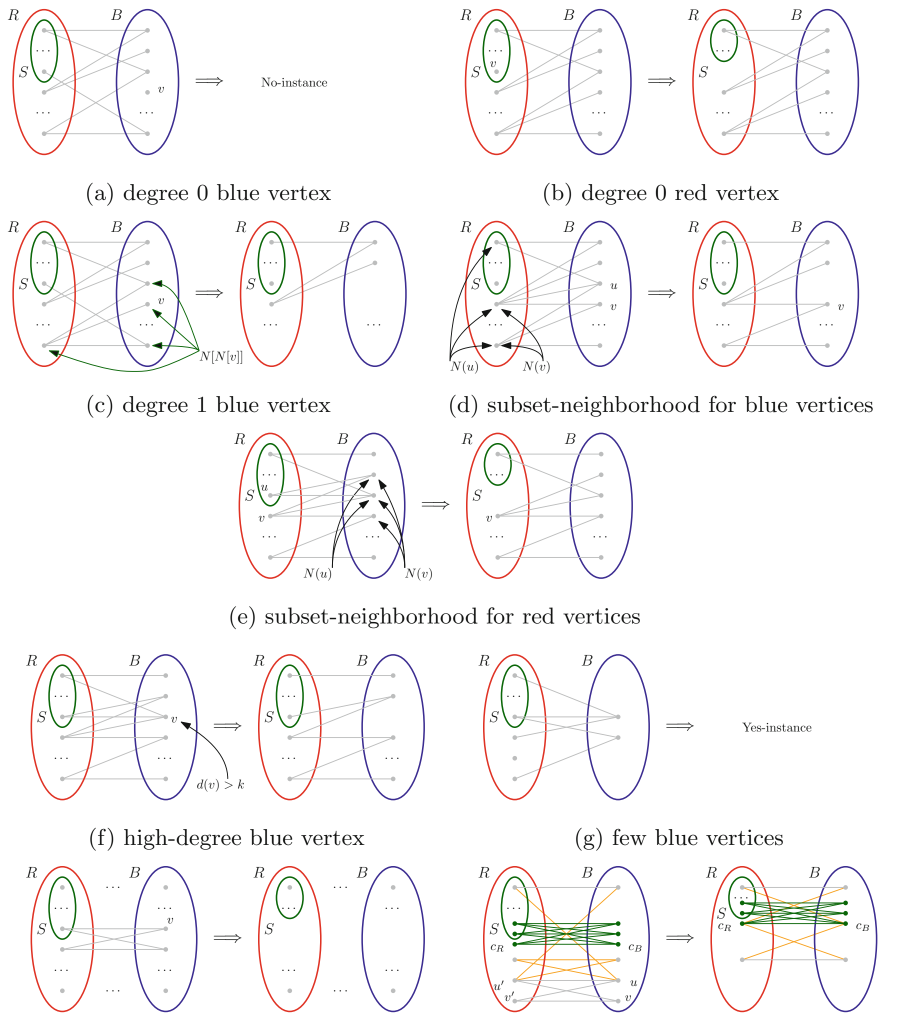
Reduction rule 3 (degree 1 blue vertex). If there exists $v \in B$ with $d(v) = 1$, then return $(G \setminus N[N[v]], k)$.

Proof (Soundness). Since vertex $v \in B$ has degree 1, its neighborhood $N(v)$ contains exactly one red vertex, say u. Vertex u cannot be part of the nonblocker. Hence, all its blue neighbors $N(u)$ will have an escape via u and we can remove them together with u from G. □

Reduction rule 4 (subset-neighborhood for blue vertices). If there exists $u, v \in B$ with $N(v) \subseteq N(u)$, then return $(G \setminus \{u\}, k)$.

Proof (Soundness). Let S be a solution for $(G \setminus \{u\}, k)$. Then there exists $x \in N(v)$ with $x \notin S$. Since, $x \in N(u)$, we have that S is a solution for (G, k) as well. □

Reduction rule 5 (subset-neighborhood for red vertices). If there exists $u, v \in R$ with $N(u) \subseteq N(v)$, then return $(G \setminus \{u\}, k - 1)$.



(a) degree 0 blue vertex

(b) degree 0 red vertex

(c) degree 1 blue vertex

(d) subset-neighborhood for blue vertices

(e) subset-neighborhood for red vertices

(f) high-degree blue vertex

(g) few blue vertices

(h) degree-2 blue vertex with degree-2 neighbors

(i) degree-2 blue vertex with one degree-2 neighbor

Fig. 3. Simple example of each reduction rule. (Color figure online)

Proof (Soundness). Let S be a size k solution for G. If neither u nor v are in S we are done. If $u \in S$, then $S \setminus \{u\}$ is a size $k-1$ solution for $G \setminus \{u\}$. If $v \in S$, then $(S \setminus \{v\}) \cup \{u\}$ is a size k solution for G as well. □

Reduction rule 6 (high-degree blue vertex). If every blue vertex has degree at least 2 and there exists $v \in B$ with $d(v) > k$, then return $(G \setminus \{v\}, k)$.

Reduction rule 7 (few blue vertices). If $|B| \leq |R| - k$, then return Yes.

Proof (Soundness). For each blue vertex select an arbitrary red neighbor and add it to a set D. Then $R \setminus D$ is a red-blue nonblocker and $|B| \leq |R| - k$ guarantees that $|R \setminus D| \geq k$. □

We now consider the Red-Blue Nonblocker variant where we are guaranteed that the input graph is symmetric and show that this problem is FPT.

Symmetric Red-Blue Nonblocker	Parameter: k
Input:	Symmetric bipartite graph $G = (R \uplus B, E)$, integer k
Question:	Is there a set $S \subseteq R$ with $\|S\| \geq k$ such that for each vertex $v \in B$, $N(v) \not\subseteq S$?

Theorem 2. Symmetric Red-Blue Nonblocker *is FPT and has a kernel with at most* $10(k + 2)/3$ *vertices.*

Proof. Let $(G = (R \uplus B, E), k)$ be an instance of Symmetric Red-Blue Nonblocker. The algorithm works as follows. First, the algorithm adds a connected component G_{cat} that is a complete bipartite graph with 3 vertices in B and 3 vertices in R. Denote by $c_R \in R$ and $c_B \in B$ two arbitrary distinguished vertices from this new component. The parameter k is increased by 2. The algorithm then exhaustively applies Reduction rule 3, as well as two new reduction rules specific to the symmetric case:

Reduction rule 8 (degree 2 blue vertex with degree-2 neighbors). If $(G = (R \uplus B, E), k)$ is an instance of Symmetric Red-Blue Nonblocker, such that there exists a vertex $v \in B$ with degree 2 and both neighbors of v have degree 2, then return $(G - N[N[v]], k - 1)$.

Reduction rule 9 (degree 2 blue vertex with one degree-2 neighbor). If $(G = (R \uplus B, E), k)$ is an instance of Symmetric Red-Blue Nonblocker, and there is a vertex $v \in B$ with degree 2 and a neighbor u' with degree at least 3, then return $(G', k - 1)$, where G' is obtained from G by deleting v and its neighbor $v' \neq u'$, merging u' with c_R and merging the vertex $u \in N(v') \setminus \{v\}$ with c_B.

If, in the resulting instance, there is a connected component C on at most 14 vertices, compute the size k_C of a largest red-blue nonblocker in this component, remove C and decrease k by k_C. If $|R| > 5k/3$, then answer Yes. Otherwise, we have a kernel with at most $10k/3$ vertices.

We will now argue the correctness of this algorithm. The first step of the algorithm adds G_{cat}, a complete bipartite graph $K_{3,3}$, as a new component and increases k by 2. It contains two distinguished vertices, $c_R \in R$ and $c_B \in B$. We observe that G_{cat} is symmetric and its largest red-blue nonblocker has size 2. We also observe that c_R and c_B are the only vertices in G_{cat} that may get additional neighbors due to reduction rules. Therefore, there exists a largest red-blue nonblocker that does not contain c_R but both of the other new vertices that were added to R. We remark that the pair (c_R, c_B) plays the same role here as the catalyst vertex in [3], i.e., a vertex that will not be added to the nonblocker and whenever we decide for some other pair of vertices that they will not be added to the nonblocker, we merge them with (c_R, c_B). We have already argued the correctness of Reduction rule 3. Moreover, it preserves symmetry since it removes the degree-1 blue vertex and its degree-1 red neighbor. We will now argue the correctness of the additional reduction rules.

Proof (Soundness of Reduction rule 8). Since G is symmetric, $N[N[v]]$ is a connected component isomorphic to a cycle of length 4, whose largest red-blue nonblocker has size 1. □

Proof (Soundness of Reduction rule 9). First, observe that every symmetry relation maps $v \in B$ to $v' \in R$ and $u \in B$ to $u' \in R$. Therefore, deleting v and v' and merging u' with c_R and merging u with c_B gives a symmetric graph.

Let S be a red-blue nonblocker for G of size at least k. W.l.o.g., we may assume that $c_R \notin S$, otherwise modify S by removing c_R from S and adding the other two red vertices from G_{cat} to S. We observe that $|S \cap \{v', u'\}| \leq 1$ since v needs an escape. Therefore, $S \setminus \{v', u'\}$ is a red-blue nonblocker for G' of size at least $k-1$. On the other hand, assume G' has a red-blue nonblocker S' of size at least $k-1$. W.l.o.g., assume $c_R \notin S'$. Then, $S' \cup \{v'\}$ is a red-blue nonblocker for G of size at least k since v can escape through u'. □

If no reduction rule applies any more, then G is a symmetric graph with minimum degree at least 3. Assume it does not have a connected component on at most 14 vertices. Let ϕ be a symmetry relation for G. Given ϕ, one can reverse the transformation from Nonblocker into equivalent instances of Red-Blue Nonblocker that was described in Sect. 1. We can construct an equivalent instance $H = (V, F)$ of Nonblocker as follows. For every $v_R \in R$ we create a new vertex $v \in V$. Vertices $u, v \in V$ are adjacent in H if there exists $(u_R, u_B) \in \phi$ such that $v_R u_B \in E$. Since G had minimum degree ≥ 3, the obtained graph H has minimum degree ≥ 2. Moreover, (H, k) is a yes-instance for Nonblocker iff (G, k) is a yes-instance for Red-Blue Nonblocker.

We now use the same strategy as Dehne et al. [3]. Namely, McCuaig and Shepherd [14] showed that, every connected graph with $n > 7$ vertices has a dominating set of size at most $2n/5$, and therefore a nonblocker of size at least $3n/5$. Now, if $|R| > 5k/3$, then the number of vertices of H is $|V| > 5k/3$ and it has a nonblocker of size at least $3|V|/5 > k$. Therefore, G has a red-blue nonblocker of size at least k and we have a yes-instance. □

Corollary 1. Red-Blue Nonblocker *can be solved in time* $1.8982^k k^{O(1)} + n^{O(1)}$.

Proof. An algorithm by Iwata [11] solves red-blue nonblocker in time $O(1.4689^{|R|})$ for symmetric instances, and it does not need access to a symmetry relation. Our polynomial-time kernelization algorithm from Theorem 2 gives an equivalent instance with $|R| \leq 5(k+2)/3$. Using Iwata's algorithm for this instance gives the desired running time. □

ALMOST SYMMETRIC RED-BLUE NONBLOCKER	Parameter: $k + \lvert D_R\rvert + \lvert D_B\rvert$
Input:	Bipartite graph $G = (R \uplus B, E)$, integer k, vertex sets $D_R \subseteq R$ and $D_B \subseteq B$ such that $H = G - (D_R \cup D_B)$ is symmetric, and symmetry relation ϕ of H
Question:	Is there a set $S \subseteq R$ with $\lvert S\rvert \geq k$ such that for each vertex $v \in B$, $N(v) \not\subseteq S$?

Theorem 3. ALMOST SYMMETRIC RED-BLUE NONBLOCKER *is FPT and can be solved in time* $O^*(1.8982^{k+|D_R|} \cdot k^{|D_B|})$.

Proof. Let $(G = (R \uplus B, E), k, D_R, D_B, \phi)$ be an instance of ALMOST SYMMETRIC RED-BLUE NONBLOCKER. In a first step we will extend the graph in order to amend the asymmetry caused by the additional red vertices D_R. For every $x \in D_R$ we add a copy of the following subgraph to G resulting in a new graph $G' = (R' \uplus B', E')$. Let $x', r_x, r'_x, b_x, b'_x \notin R \uplus B$ be new vertices, where r_x and r'_x are added to R' and x', b_x, and b'_x are added to B'. Additionally, we add the following new edges to E': xx', xb_x, r_xx', r_xb_x, $r_xb'_x$, r'_xb_x, and $r'_xb'_x$.

Now G' has a size $k+|D_R|$ nonblocker if and only if G has a size k nonblocker. To prove this, we show that this holds for $|D_R| = 1$, say $D_R = \{x\}$. Using the same argument inductively results in the above statement. Assume S is a size k nonblocker of G. Then $S \cup \{r'_x\}$ is a size $k + 1$ nonblocker of G, since the neighborhood of B remained the same and the new blue vertices x', b_x, b'_x have neighbor $r_x \notin S$. For the other direction, assume S is a size $k + 1$ nonblocker of G'. Then either r_x or r'_x (or both) is not part of S, since otherwise b'_x does not have a neighbor outside S. Hence, $|S \cap R| \geq k$ and $S \cap R$ is a nonblocker of G since the neighborhood of B remained unchanged by our transformation. This concludes, that searching for a size $k + |D_R|$ nonblocker of G' leads to an equivalent instance where the only asymmetry left is due to additional blue vertices D_B. We extend the symmetry relation ϕ to ϕ' which accounts for the newly introduced vertices.

For the second step, let x be an arbitrary vertex in D_B. If $|N(x)| > k$ we can use Reduction rule 6 to remove x from G'. Hence, assume $|N(x)| \leq k$. Since at least one of its neighbors has to be outside of the nonblocker, we will brute force over all $\leq k$ candidates $y \in N(x)$ and successively mark them as not belonging to the nonblocker as well as remove x from G'. In total we have $k^{|D_B|}$ possible combinations of marked red vertices and G' is symmetric in all cases. Hence, we can use the symmetry relation ϕ' to transform our instance $(G', k + |D_R|)$ into an equivalent instance $(H', k + |D_R|$ of NONBLOCKER while preserving the fact that some of the vertices are marked as not belonging to the nonblocker. Now we can use the algorithm by Dehne et al. [3], which is capable of dealing with marked vertices and decides NONBLOCKER in time $1.8982^k n^{O(1)}$ for parameter value k if the search algorithm is replaced with Iwata's algorithm [11]. Hence, in total we can solve ALMOST SYMMETRIC RED-BLUE NONBLOCKER in time $1.8982^{k+|D_R|} \cdot k^{|D_B|} n^{O(1)}$. □

3 Not Too Many Blue Vertices

As stated in the introduction it is known that the Red-Blue Nonblocker problem is $W[1]$-hard [6]. But what if the number of blue vertices is linear in the number of red vertices, or even upper bounded by $\ell \cdot |R|$ for an additional parameter ℓ? That is, we consider the Red-Blue Nonblocker problem augmented with the parameter $\ell = |B|/|R|$. Somewhat surprisingly we will show that the problem is FPT if the instance has no blue vertices of degree 1, otherwise it is $W[1]$-hard for the parameter $\ell + k$.

First consider the parameterization of Red-Blue Nonblocker by $\ell + k$, and there is no restriction on the degree of the vertices. We can easily show that this version is $W[1]$-hard by a reduction from the Red-Blue Nonblocker problem with parameter k. Given an instance $(G = (R \uplus B, E), k)$, simply add $|B|$ vertices $r_1, \ldots, r_{|B|}$ to R and $|B|$ vertices $b_1, \ldots, b_{|B|}$ to B and add the $|B|$ edges $(r_1, b_1), \ldots, (r_{|B|}, b_{|B|})$ to E. This does not change the size of the nonblocker but gives $\ell \leq 2$. Thus, we have the following result.

Proposition 1. *The* Red-Blue Nonblocker *is $W[1]$-hard for parameter $\ell + k$.*

The above hardness proof relies heavily on the fact that there are a large number of vertices with degree 1. What if G has no blue vertex of degree 1? We next show that the Red-Blue Nonblocker problem is FPT if we augment the parameter by ℓ and do not allow any vertex in B to have degree 1.

Consider the following algorithm for the Red-Blue Nonblocker problem parameterized by $k + \ell$ when no vertex in B has degree 1. Given such an instance $(G = (R \uplus B, E), k)$ with $|B| = \ell \cdot |R|$, the algorithm first invokes Reduction rule 1. If $|R| \geq (2\ell + 1)k$ then report $(G = (R \uplus B, E), k)$ as a yes-instance. Otherwise, if $|R| < (2\ell + 1)k$, solve the Red-Blue Nonblocker problem by brute-force, by checking for each size-k subset of R whether it is a red-blue nonblocker.

For the correctness of the algorithm, we start with an important definition.

Definition 2. *A* red graph *of a bipartite graph $G = (R \uplus B, E)$, where each vertex in B has degree at least 2, is a graph obtained from the hypergraph $(R, \{N_G(v) : v \in B\})$ by shrinking each hyperedge to size 2 in an arbitrary way.*

Figure 4 shows an example of a bipartite graph and a red graph that it induces. Before proving the final theorem we need the following two lemmas.

Lemma 1. *If $|R| \geq (2\ell + 1)k$ then any red graph $H = (R, F)$ of $G = (R \uplus B, E)$ has an independent set of size at least k.*

Proof. Consider a red graph $H = (R, F)$ of $G = (R \uplus B, E)$. Since $|B| = \ell \cdot |R|$ the average degree of H is at most 2ℓ. Turán [16] (see, e.g., [10] for a proof in English) showed that for any graph $\mathfrak{G}$, an independent set of size $\frac{n}{\overline{d}+1}$ can be found in linear time, where n is the number of vertices and $\overline{d}$ the average degree of $\mathfrak{G}$. Applying Turán's [16] bound on H gives that H has an independent set of size at least $|R|/(2\ell + 1)$. The statement of the lemma immediately follows. $\square$

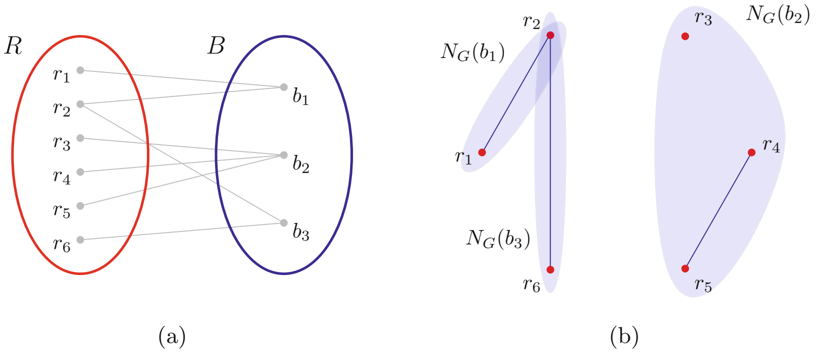


Fig. 4. Example of (a) bipartite graph G and (b) its corresponding red graph. For each hyperedge, an arbitrary edge is selected, e.g. the hyperedge induced by $N_G(b_2) = \{r_3, r_4, r_5\}$ is shrunk to the edge $\{r_4, r_5\}$ in the red graph. (Color figure online)

Lemma 2. *If S is an independent set of a red graph $H = (R, F)$ of the bipartite graph $G = (R \uplus B, E)$, then S is a red-blue nonblocker for G.*

Proof. Let $S \subseteq R$ be an independent set of H. To show that S is a red-blue nonblocker for G, it suffices to show that every blue vertex $v \in B$ has an escape, i.e., a neighbor in $R \setminus S$. We observe that $R \setminus S$ is a vertex cover of H. Recall, from the definition of red graphs, that there is a bijection between vertices in B and edges in H, and each edge in H is incident to two vertices that its corresponding blue vertex is adjacent to. Since each edge in H has an incident vertex in the vertex cover $R \setminus S$, its corresponding blue vertex has an escape in $R \setminus S$. □

Theorem 4. Red-Blue Nonblocker *is FPT for the parameter $k + \ell$ when no blue vertex has degree 1, where $\ell = |B|/|R|$.*

Proof. The running time of the algorithm is $\binom{(2\ell+1)k}{k} \cdot n^{O(1)}$.

For the correctness, first observe that Reduction rule 1 returns No if there is a blue vertex with degree 0. Thus, red graphs are well-defined in what follows. Let $H = (R, F)$ be a red graph for G. For the case where $|R| \geq (2\ell + 1)k$, by Lemma 1, H has an independent set S of size at least k. By Lemma 2, S is a red-blue nonblocker for G. Therefore, the algorithm correctly reports that G is a Yes-instance. For the case where $|R| < (2\ell + 1)k$, the correctness follows because the search is exhaustive. □

We observe that degree-1 blue vertices can be handled by Reduction rule 3, and if their number is bounded by a function of the parameter, the application of this reduction rule does not change the value of ℓ significantly. We also observe that fast Set Cover algorithms [7,11] can be used instead of the brute-force approach when $|R| < (2\ell + 1)k$. For example, we could use the polynomial-space algorithm of [11] solving the instance in time $1.4864^{0.6359|R|+0.3642|B|}n^{O(1)}$.

Note that if the average degree of the vertices in R is at most d_R then the number of vertices in B is at most $d_R \cdot |R|$. Hence, as a corollary, we get:

Corollary 2. RED-BLUE NONBLOCKER *is FPT for the parameter* $k + d_R$ *on instances where no vertex in* B *has degree* 1*, where* d_R *denotes the average degree of the vertices in* R*.*

We note that RED-BLUE NONBLOCKER is W[1]-hard for the parameter $k + d_R$ when we allow blue vertices of degree 1, since we can again add a large enough set of isolated edges to the instance to reduce d_R to a constant value.

Acknowledgments. Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048). This work received support under the ARC's Discovery Projects funding scheme (DP150101134).

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. Algorithmica **33**(4), 461–493 (2002). https://doi.org/10.1007/s00453-001-0116-5
2. Cairns, G., Mendan, S.: Symmetric bipartite graphs and graphs with loops. Discrete Math. Theor. Comput. Sci. **17**(1), 97–102 (2015). http://dmtcs.episciences.org/2119
3. Dehne, F., Fellows, M., Fernau, H., Prieto, E., Rosamond, F.: NONBLOCKER: parameterized algorithmics for MINIMUM DOMINATING SET. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 237–245. Springer, Heidelberg (2006). https://doi.org/10.1007/11611257_21
4. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: on completeness for W[1]. Theor. Comput. Sci. **141**(1&2), 109–131 (1995). https://doi.org/10.1016/0304-3975(94)00097-3
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999). https://doi.org/10.1007/978-1-4612-0515-9
6. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer, Heidelberg (2013). https://doi.org/10.1007/978-1-4471-5559-1
7. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 25:1–25:32 (2009). https://doi.org/10.1145/1552285.1552286
8. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30559-0_21
9. Grandoni, F.: A note on the complexity of minimum dominating set. J. Discrete Algorithms **4**(2), 209–214 (2006). https://doi.org/10.1016/j.jda.2005.03.002
10. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: approximating independent sets in sparse and bounded-degree graphs. Algorithmica **18**(1), 145–163 (1997). https://doi.org/10.1007/BF02523693
11. Iwata, Y.: A faster algorithm for dominating set analyzed by the potential method. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 41–54. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28050-4_4

12. Kanj, I.A., Xia, G.: When is weighted satisfiability FPT? In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 451–462. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_39
13. Lubiw, A.: Some NP-complete problems similar to graph isomorphism. SIAM J. Comput. **10**(1), 11–21 (1981). https://doi.org/10.1137/0210002
14. McCuaig, W., Shepherd, F.B.: Domination in graphs with minimum degree two. J. Graph Theory **13**(6), 749–762 (1989). https://doi.org/10.1002/jgt.3190130610
15. Ore, O.: Theory of Graphs. American Mathematical Society Colloquium Publications. American Mathematical Society, Providence (1962)
16. Turán, P.: On an extremal problem in graph theory. Matematikai és Fizikai Lapok **48**, 436–452 (1941). In Hungarian

Incremental Strong Connectivity and 2-Connectivity in Directed Graphs

Loukas Georgiadis[1], Giuseppe F. Italiano[2], and Nikos Parotsidis[2(✉)]

[1] University of Ioannina, Ioannina, Greece
loukas@cs.uoi.gr
[2] University of Rome Tor Vergata, Rome, Italy
{giuseppe.italiano,nikos.parotsidis}@uniroma2.it

Abstract. In this paper, we present new incremental algorithms for maintaining data structures that represent all connectivity cuts of size one in directed graphs (digraphs), and the strongly connected components that result by the removal of each of those cuts. We give a conditional lower bound that provides evidence that our algorithms may be tight up to a sub-polynomial factors. As an additional result, with our approach we can also maintain dynamically the 2-vertex-connected components of a digraph during any sequence of edge insertions in a total of $O(mn)$ time. This matches the bounds for the incremental maintenance of the 2-edge-connected components of a digraph.

1 Introduction

A dynamic graph algorithm aims at updating efficiently the solution of a graph problem after an update, faster than recomputing it from scratch. A dynamic graph problem is said to be *fully dynamic* if the update operations include both insertions and deletions of edges, and it is said to be *incremental* (resp., *decremental*) if only insertions (resp., deletions) are allowed. In this paper, we present new incremental algorithms for some basic connectivity problems on directed graphs (digraphs), which were recently considered in the literature [12]. Before defining the problems and stating our bounds, we need some definitions.

Let $G = (V, E)$ be a digraph. G is *strongly connected* if there is a directed path from each vertex to every other vertex. The *strongly connected components* (in short *SCCs*) of G are its maximal strongly connected subgraphs. Two vertices $u, v \in V$ are *strongly connected* if they belong to the same SCC of G. An edge (resp., a vertex) of G is a *strong bridge* (resp., a *strong articulation point*) if its removal increases the number of SCCs in the remaining graph. Let G be strongly connected: G is *2-edge-connected* (resp., *2-vertex-connected*) if it has no strong bridges (resp., no strong articulation points). Two vertices $u, v \in V$ are said to be *2-edge-connected* (resp., *2-vertex-connected*), denoted by $u \leftrightarrow_{2e} v$ (resp., $u \leftrightarrow_{2v} v$), if there are two edge-disjoint (resp., internally vertex-disjoint) directed paths from u to v and two edge-disjoint (resp., internally vertex-disjoint)

Full version of this paper is available in https://arxiv.org/abs/1802.10189.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 529–543, 2018.
https://doi.org/10.1007/978-3-319-77404-6_39

directed paths from v to u. (Note that a path from u to v and a path from v to u need not be edge-disjoint or internally vertex-disjoint). A *2-edge-connected component* (resp., a *2-vertex-connected component*) of a digraph $G = (V, E)$ is defined as a maximal subset $B \subseteq V$ such that $u \leftrightarrow_{2\mathrm{e}} v$ (resp., $u \leftrightarrow_{2\mathrm{v}} v$) for all $u, v \in B$. Given a digraph G, we denote by $G \setminus e$ (resp., $G \setminus v$) be the digraph obtained after deleting edge e (resp., vertex v) from G.

Let $G = (V, E)$ be a strongly connected graph. In very recent work [12], we presented an $O(n)$-space data structure that, after a linear-time preprocessing, is able to answer in asymptotically optimal (worst-case) time all the following queries on a static digraph:

– Report in $O(1)$ time the total number of SCCs in $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v) in G.
– Report in $O(1)$ time the size of the largest and of the smallest SCCs in $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v) in G.
– Report in $O(n)$ time all the SCCs of $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v).
– Test in $O(1)$ time whether two query vertices u and v are strongly connected in $G \setminus e$ (resp., $G \setminus w$), for any query edge e (resp., vertex w).
– For any two query vertices u and v that are strongly connected in G, report all edges e (resp., vertices w) such that u and v are not strongly connected in $G \setminus e$ (resp., $G \setminus w$) in time $O(k + 1)$, where k is the number of reported edges (resp., reported vertices).

As pointed out in [12,21], this data structure finds applications in many areas, including computational biology [13,20], social network analysis [18,25], network resilience [23] and network immunization [3,5,19]. A dynamic version of this data structure can be used to monitor critical edges and vertices, i.e., edges and vertices whose removal disrupts an underlying graph which evolves over time.

Our Results. We show a conditional lower bound for the fully dynamic version of this problem. More specifically, let $G = (V, E)$ be a digraph with n vertices that undergoes m edge updates from an initially empty graph. We prove that any fully dynamic algorithm that can answer all of the queries considered here requires either $\Omega(m^{1-o(1)})$ amortized update time, or $\Omega(m^{1-o(1)})$ query time, unless the Strong Exponential Time Hypothesis [15,16] is false.

Motivated by this hardness result, we focus on the incremental version of this problem. We present an incremental version of the data structure introduced in [12], which can be maintained throughout a sequence of edge insertions. In particular, we show how to maintain a digraph G undergoing edge insertions in a total of $O(mn)$ time, where n is the number of vertices and m the number of edges after all insertions, so that all the queries we consider can be answered in asymptotically optimal (worst-case) time after each insertion. As an additional result, with our approach we can also maintain the 2-vertex-connected components of a digraph during any sequence of edge insertions in a total of $O(mn)$ time. After every insertion we can test whether two query vertices are 2-vertex-connected and, whenever the answer is negative, produce a separating vertex (or an edge) for the two query vertices. This matches the bounds for the incremental maintenance of the 2-edge-connected components of a digraph [11].

Before our work, no algorithm for all those problems was faster than recomputing the solution from scratch after each edge insertion, which yields a total of $O(m^2)$ update time. Our algorithms improve substantially over those bounds. In addition, we show a conditional lower bound for the total update time of an incremental data structure that can answer queries of the form "are u and v strongly connected in $G \setminus e$", where $u, v \in V$, $e \in E$. In particular, we prove that the existence of a data structure that supports those queries with total update time $O((mn)^{1-\epsilon})$ (for some constant $\epsilon > 0$) refutes the online Matrix-vector multiplication conjecture [14]. Therefore, a polynomial improvement of our bound would lead to a breakthrough.

Related Work. Many efficient algorithms for several dynamic graph problems have been proposed in the literature (see, e.g., the survey in [7]). Dynamic problems on digraphs are known to be harder than on undirected graphs [1]: indeed, most of the dynamic algorithms on undirected graphs have polylog update bounds, while dynamic algorithms on digraphs have higher polynomial update bounds. In [9], the decremental version of the data structure considered in this paper is presented. The total time and space required to maintain decrementally the data structure is $O(mn \log n)$ and $O(n^2 \log n)$, respectively: here m is the number of edges in the initial graph. In [11] we presented an incremental algorithm that maintains the 2-edge-connected components of a directed graph with n vertices through any sequence of edge insertions in a total of $O(mn)$ time, where m is the number of edges after all insertions. We remark that our incremental algorithms and techniques are substantially different from the decremental ones in [9], and that 2-vertex connectivity in digraphs is much more difficult than 2-edge connectivity, since it is plagued with several degenerate special cases, which are more cumbersome to deal with.

Our Technical Contributions. Our main contribution is to dynamize the recent data structure in [12], which hinges on two main building blocks: dominator trees and loop nesting trees (which will be reviewed in Sect. 2). While it is known how to maintain efficiently dominator trees in the incremental setting [10], the incremental maintenance of loop nesting trees is a challenging task. Indeed, loop nesting trees are heavily based on depth-first search, and maintaining efficiently a dfs tree of a digraph under edge insertions has been an elusive goal: no efficient solutions are known up to date, and incremental algorithms are available only in the restricted case of DAGs [8]. To overcome these inherent difficulties, we manage to define a new notion of strongly connected subgraphs of a digraph, which is still relevant for our problem and is independent of depth first search. This new notion is based on some specific nesting loops, which define a laminar family. One of the technical contributions of this paper is to show how to maintain efficiently this family of nesting loops during edge insertions. We believe that this result might be of independent interest, and perhaps it might shed further light to the incremental dfs problem on general digraphs.

2 Dominators and Loops

We assume that the reader is familiar with standard graph terminology, as contained for instance in [6]. Given a rooted tree, we denote by $T(v)$ the set of descendants of v in T. Given a digraph $G = (V, E)$, and a set of vertices $S \subseteq V$, we denote by $G[S]$ the subgraph induced by S. Moreover, we use $V(S)$ and $E(S)$ to refer to the vertices of S and to the edges adjacent to S, respectively. The *reverse digraph* of G, denoted by $G^R = (V, E^R)$, is obtained by reversing the direction of all edges. A *flow graph* F is a directed graph (digraph) with a distinguished start vertex $s \in V(F)$, where all vertices in $V(F)$ are reachable from s in F. We denote by G_s the subgraphs of G induced by the vertices that are reachable from s; that is, G_s is a flow graph with start vertex s. Respectively, we denote by G_s^R the subgraphs of G^R induced by the vertices that are reachable from s. If G is strongly connected, all vertices are reachable from s and reach s, so we can view both G and G^R as flow graphs with start vertex s.

Dominator trees. A vertex v is a *dominator* of a vertex w (v *dominates* w) if every path from s to w contains v. The dominator relation in G can be represented by a tree rooted at s, the *dominator tree* D, such that v dominates w if and only if v is an ancestor of w in D. See Fig. 1. We denote by $dom(w)$ the set of vertices that dominate w. Also, we let $d(w)$ denote the parent of a vertex w in D. Similarly, we can define the dominator relation in the flow graph G_s^R, and let D^R denote the dominator tree of G_s^R, and $d^R(v)$ the parent of v in D^R. The dominator tree of a flow graph can be computed in linear time, see, e.g., [2,4]. An edge (u, v) is a *bridge* of a flow graph G_s if all paths from s to v include (u, v).[1] Let s be an arbitrary start vertex of G. As shown in [17], an edge $e = (u, v)$ is strong bridge of G if and only if it is either a bridge of G_s or a bridge of G_s^R. As a consequence, all the strong bridges of G can be obtained from the bridges of the flow graphs G_s and G_s^R, and thus there can be at most $2(n-1)$ strong bridges overall. After deleting from the dominator trees D and D^R respectively the bridges of G_s and G_s^R, we obtain the *bridge decomposition* of D and D^R into forests $\mathcal{D}$ and $\mathcal{D}^R$. Throughout the paper, we denote by D_u (resp., D_u^R) the tree in $\mathcal{D}$ (resp., $\mathcal{D}^R$) containing vertex u, and by r_u (resp., r_u^R) the root of D_u (resp., D_u^R).

Updating the dominator tree after an edge insertion. We briefly review the algorithm from [10] that updates the dominator tree of a flow graph G_s after an edge insertion. Let G_s be a flow graph with start vertex s. Let (x, y) be the edge to be inserted. Let D be the dominator tree of G_s before the insertion; we let D' be the dominator tree of G'_s. In general, for any function f on V, we let f' be the function after the update. We say that vertex v is *D-affected* by the update if $d(v)$ (its parent in D) changes, i.e., $d'(v) \neq d(v)$. We let $nca_D(x, y)$ denote the nearest common ancestor of x and y in the dominator tree D. Then, every D-affected vertex v becomes a child of $nca_D(x, y)$ in D', i.e., $d'(v) = nca_D(x, y)$ [22].

[1] Throughout the paper, we use the term *bridge* to refer to a bridge of a flow graph and the term *strong bridge* to refer to a strong bridge in the original graph.

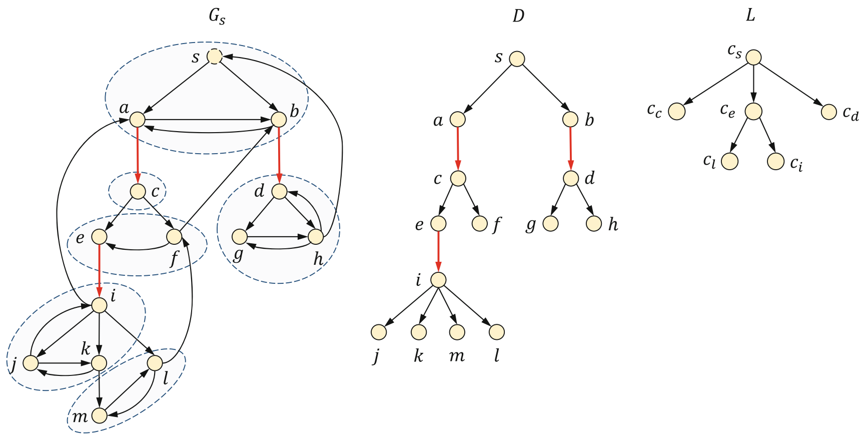


Fig. 1. A flow graph G_s and its dominator tree D and hyperloop nesting tree L. The grouped vertices in both G_s and L represent the auxiliary components of G_s.

Hence, for any vertex u that is not a descendant of $nca_D(x, y)$ in D' we have $dom(u) = dom'(u)$ and $D'(u) = D(u)$. Moreover, if $(d(u), u)$ was a bridge in G_s, then it remains a bridge in G'_s. We say that a vertex is *D-scanned* if it is a descendant of a D-affected vertex after an edge insertion. There are two key ideas behind the incremental dominators algorithm that result to an $O(mn)$ total update time. First, the algorithm updates D' in time proportional to number of the edges incident to D-scanned vertices. Second, after an edge insertion, all D-scanned vertices decrease their depth in D' by at least one.

Loop nesting forests. Let G be a digraph, and G_s the flow graph with arbitrary start vertex s. A *loop nesting forest* represents a hierarchy of strongly connected subgraphs of G_s [24], defined with respect to a dfs tree T of G_s, rooted at s: for any vertex u, $loop(u)$ is the set of all descendants x of u in T such that there is a path from x to u in G containing only descendants of u in T. Any two vertices in $loop(u)$ reach each other, therefore $loop(u)$ induces a strongly connected subgraph of G. In the *loop nesting forest* H of G_s, with respect to T, the parent of any vertex v, denoted by $h(v)$, is the nearest proper ancestor u of v in T such that $v \in loop(u)$ if there is such a vertex u, and null otherwise. Then $loop(u)$ is the set of all descendants of vertex u in H, which we will also denote as $H(u)$. A loop nesting forest can be computed in linear time [4,24]. When G is strongly connected, each vertex is contained in a loop, and H is a tree, rooted at s. Therefore, we refer to H as the *loop nesting tree* of G_s.

Auxiliary components. Let G_s be a flow graph and D and $\mathcal{D}$ be the dominator tree and the bridge decomposition of G_s, respectively. Let $e = (u, v)$ be a bridge of the flow graph G_s. We say that an SCC C in $G[D(v)]$ is an *e-dominated component* of G. We also say that $C \subseteq V$ is a *bridge-dominated component* if it is an e-dominated component for some bridge e: bridge-dominated components form a laminar family [11]. An *auxiliary component* of G_s is a maximal subset of vertices $C \cap D_v$ such that C is a subset of a $(d(r_v), r_v)$-dominated component. Each auxiliary component C is represented by an arbitrarily chosen vertex $u \in C$, which we call the *canonical vertex* of C. For each vertex $v \in C$, we refer to the canonical vertex of C by c_v ($c_v = v$ if v is a canonical vertex).

3 Hyperloop Nesting Forest

We now introduce the new notion of *hyperloop nesting forest*, which, differently from loop nesting forest, can be maintained efficiently during edge insertions. Given a canonical vertex $v \neq c_s$, we define the *hyperloop of* v, and denote it by $hloop(v)$, as the set of canonical vertices that are in the same $(d(r_v), r_v)$-dominated component as v. As a special case, all canonical vertices that are strongly connected to s are in the hyperloop $hloop(c_s)$. It can be shown that hyperloops form a laminar family of subsets of V, with respect to the start vertex s: for any two canonical vertices u and v, $hloop(u)$ and $hloop(v)$ are either disjoint or nested (i.e., one contains the other). This allows us to define the *hyperloop nesting forest* L of G_s as follows. The parent $\ell(v)$ of a canonical vertex v in L is the (unique) canonical vertex u, $u \notin D(r_v)$, with the largest depth in D, such that $v \in hloop(u)$. If there is no vertex $u \notin D(r_v)$, such that $v \in hloop(u)$, then $\ell(v) = \emptyset$; note that in this case v is not strongly connected to s as well. See Fig. 1. Then, $hloop(u)$ is the set of all descendants of a canonical vertex u in L, which we will also denote as $L(u)$. Similarly to the loop nesting forest, the hyperloop nesting forest of a strongly connected digraph is a tree.

Consider a fixed choice of the canonical vertices of the auxiliary components in G_s. Then, the hyperloop nesting forest L of G_s is unique. Moreover, all the ancestors of a canonical vertex in L have distinct levels. We can show that hyperloop nesting forest L can be obtained from the loop nesting forest H by contracting all the vertices of each auxiliary component into their canonical vertex. This yields immediately a linear-time algorithm to compute the hyperloop nesting forest of a flow graph G_s: we first compute a loop nesting forest H of G_s [4,24] and then contract each vertex v to c_v in H.

3.1 Updating the Hyperloop Nesting Forest After an Edge Insertion

Let G be a directed graph and let G_s be the flow graph of G with start vertex s, so D and L are rooted at s and c_s, respectively. For simplicity, we assume that all vertices of G are reachable from s. If this is not true, then we can simply recompute D and L from scratch, in linear time, every time a vertex becomes reachable from s after an edge insertion. Since there can be at most $n - 1$ such events, the total running time for these recomputations is $O(mn)$. Throughout the sequence of edge insertions, we maintain as additional data structures only the dominator tree D (with the algorithm in [10]), the bridge decomposition and the auxiliary components of G_s (with the algorithm in [11]).

Initialization and restarts. To initialize the algorithm, we compute the dominator tree D, bridge decomposition and auxiliary components, and also the hyperloop nesting forest L of G_s in linear time. After the first initialization, in some special cases we initialize our algorithm again, in order to simplify the analysis. We call this a *restart.* We restart our algorithm whenever a bridge $e = (u, v)$ of G_s is canceled after the insertion of a new edge (x, y) but we still have $d'(v) = u$, i.e., (u, v) is no longer a strong bridge in G but the parent of u in the dominator tree D does not change. In this case, we say that the bridge $e = (u, v)$ is *locally canceled.* This is a difficult case to analyze: the incremental dominators algorithm does not spend any time, since there are no D-affected vertices, while the bridge decomposition and the auxiliary components of G_s might change. Fortunately, there can be at most $O(n)$ locally canceled bridges throughout a sequence of edge insertions [11]. Hence, we restart our algorithm at most $O(n)$ times. Consequently, the total time spent in restarts is $O(mn)$.

High-level overview of the update. Let (x, y) be the new edge to be inserted. As in Sect. 2, for any function f, we let f' denote the function after the update, e.g., we denote by $\ell'(v)$, the parent of a canonical vertex v in the hyperloop nesting forest, after the insertion of (x, y), and by L' the resulting hyperloop nesting forest. We say that a canonical vertex v is *L-affected* if $\ell'(v') \neq c'_{\ell(v)}$, i.e., when the parent of v' in L' is not in the same auxiliary component as $\ell(v)$ (the parent of v in L). After the insertion of a new edge (x, y), if not involved in a restart, our algorithm performs the following updates:

(1) Compute the new dominator tree D', the corresponding bridge decomposition $\mathcal{D}'$, and the new auxiliary components.
(2) Compute $\ell'(v)$ for the D-scanned canonical vertices $v \in D'_y$.
(3) Compute $\ell'(v)$ for the D-scanned canonical vertices $v \notin D'_y$.
(4) Compute $\ell'(v)$ for the L-affected canonical vertices v that are not D-scanned.

As already mentioned, we compute (1) within our claimed bounds [10,11]. To complete the algorithm, it remains to show how to update efficiently the parents in the hyperloop nesting forest of the D-scanned and the L-affected vertices, which is a non-trivial task. Before giving the details of our algorithm, we observe that no canonical vertex $v \notin D'(r'_y)$ is L-affected.

Updating the D-scanned vertices. Let S be the set of D-scanned vertices containing also the D-affected vertices. After the insertion of the edge (x, y), all the D-affected vertices become children of $nca_D(x, y)$ in D' [22]. In this section we deal with the update of the parent in the hyperloop nesting forest $\ell'(v')$, for all canonical vertices $v \in S$. Given a vertex u, we define its level, denoted by $level(u)$, to be the number of bridges (v, w) of G_s such that w is an ancestor of u in D. In other words, the level of u equals the number of bridges that appear in all paths from s to u in G_s. From now on, in order to simplify the notation, we assume without loss of generality that $v = c_v$ for any vertex of interest v; we also denote c'_v by v'. After the insertion of the edge (x, y) only a subset of the ancestors in L' of an L-affected canonical vertex v changes. In particular, we can show that the ancestors w of v in L such that $w \notin D'(r'_y)$ remain ancestors of v' in L'. However, the insertion of (x, y) might create a new path from v to a canonical vertex z such that $v \in D'(r'_z)$, containing only vertices in $D'(r'_z)$. In such a case, z' becomes an ancestor of v' in L'. We now compute $\ell'(v')$ for each D-scanned vertex v that is in the same tree of the canonical decomposition of D' with y, including y'.

Lemma 1. *For every D-scanned vertex $v \in D'_y$ it holds $\ell'(v') = w'$, where w is the nearest ancestor of v in L such that $w \notin D'(r'_y)$. If there is no such vertex w, then $\ell'(v') = \emptyset$.*

Next, we deal with the canonical vertices $v \in S \setminus D'_y$. We begin with the computation of $\ell'(v')$, for the canonical vertices v' in S for which $\mathit{level}'(\ell'(v')) > \mathit{level}'(r'_y)$, that is, their new parent in L is in $D'(r'_y) \setminus D'_y$. Let $G_{scanned}$ be the graph induced by the D-scanned vertices, and let $H_{scanned}$ be the loop nesting forest rooted at y of $G_{scanned}$. (Note that y reaches all vertices in $G_{scanned}$.) By contracting every vertex v into c'_v in $H_{scanned}$, we obtain a forest $\tilde{H}$. Let $\tilde{h}(v)$ be the parent in $\tilde{H}$ of a canonical vertex $v \in S \setminus D'_y$. We can show that if $\tilde{h}(v) \in S \setminus D'_y$, then $\tilde{h}(v)$ is the parent of v' in L', i.e., $\ell'(v') = \tilde{h}(v)$. Finally, for the vertices $v \in S \setminus D'_y$ for which $\mathit{level}'(\ell'(v')) \leq \mathit{level}'(r'_y)$, we compute $\ell'(v')$ according to the following lemma.

Lemma 2. *Let $v \notin D'_y$ be a D-scanned vertex such that $\mathit{level}'(\ell'(v')) \leq \mathit{level}'(r'_y)$. If v has a path to y in $G'[D'(r'_y)]$, then $\ell'(v') = y'$ in L'. Otherwise, $\ell'(v') = w'$, where w is the nearest ancestor of v in L such that $\mathit{level}'(w) \leq \mathit{level}'(r'_y)$. If there is no such vertex w, then $\ell'(v') = \emptyset$.*

Using the above lemmas, we update the parent in L' of the D-scanned vertices S in time that is linear in the size of $V(S)$ and $E(S)$. Note that in Lemma 2 we need to determine whether a D-scanned vertex v has a path to y in $G'[D'(r'_y)]$. We compute this information in time $O(|E(S)|)$ for all D-scanned vertices.

Updating the L-affected vertices that are not D-scanned. The next lemma suggests that we can find all the L-affected vertices via a backward traversal from y visiting all vertices in $G'[D'(r'_y)]$ that have a path to y. A straightforward execution of this traversal, however, requires $O(m)$ time which we cannot afford. Later, we show how to speed up this process by exploiting some key properties of the L-affected vertices.

Lemma 3. *For every L-affected vertex v that is not D-scanned, every path from v to $\ell'(v')$ in $G'[D'(r'_{\ell'(v')})]$ contains (x, y). Moreover, v has a path to x in $G[D'(r'_y)] = G'[D'(r'_y)] \setminus (x, y)$.*

Recall that only vertices in $D'(r'_y)$ can be L-affected. Hence, we only need to consider the vertices that are in $D'(r'_y)$ and are not D-scanned.

Lemma 4. *Let (x, y) be the newly inserted edge. The canonical vertex v' of a vertex $v \in D'(r'_y)$ such that v is not D-scanned and v has a path to x in $G'[D'(r'_y)]$, changes its parent $\ell'(v')$ as follows: (see Fig. 2 for an illustration of the different cases)*

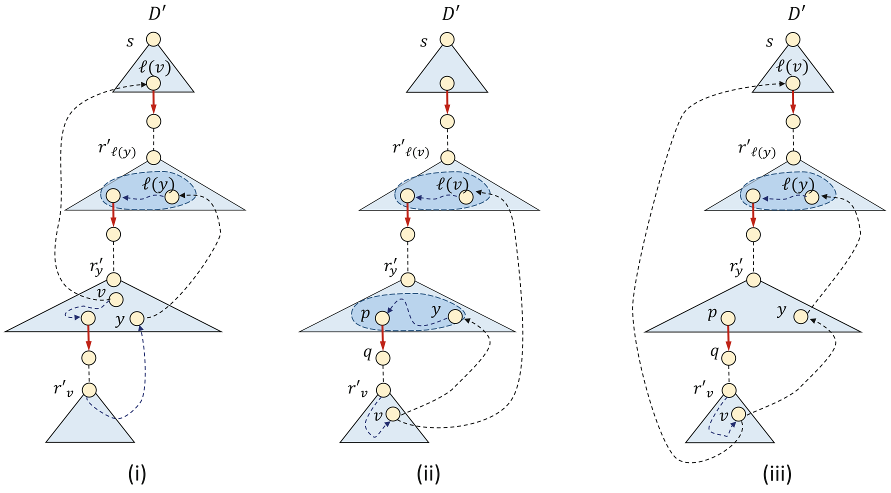


Fig. 2. A demonstration of the different cases in Lemma 4. (i) Case (1) where $v \in D'_y$ and $level'(\ell(v)) < level'(\ell'(y'))$. Here we have $\ell'(v') = \ell'(y')$. (ii) Case (2.1) where $v \notin D'_y, c'_p = y'$ and $level'(\ell(v)) < level'(c'_p)$. Now we have $\ell'(v') = c'_p$. (iii) Case (2.2) where $v \notin D'_y, c'_p \neq y'$ and $level'(\ell(v)) < level'(\ell'(y'))$. In this case $\ell'(v') = \ell'(y')$.

(1) Case $v \in D'_y$: if $level'(\ell(v)) < level'(\ell'(y'))$ or $\ell(v) = \emptyset$ then $\ell'(v') = \ell'(y')$. Otherwise, $\ell'(v') = c_{\ell(v)}$.
(2) Case $v \notin D'_y$: let (p, q) the strong bridge such that $p \in D'_y$ and q is an ancestor of v in D'.
(2.1) Case $c'_p = y'$: if $level'(\ell(v)) < level'(c'_p)$ or $\ell(v) = \emptyset$, then $\ell'(v') = c'_p$. Otherwise, $\ell'(v') = c'_{\ell(v)}$.
(2.2) Case $c'_p \neq y'$: if $level'(\ell(v)) < level'(\ell'(y'))$ or $\ell(v) = \emptyset$, then $\ell'(v') = \ell'(y')$. Otherwise, $\ell'(v') = c'_{\ell(v)}$.

Lemma 4 shows how to determine the new parent in L' of each canonical vertex $v \in D'(r'_y)$ that is L-affected but not D-scanned. The most challenging computation is to determine which vertices have a path to x in $G'[D'(r'_y)]$. We show how to compute efficiently those vertices by executing a backward traversal: this runs in time proportional to the sum of the degrees of the L-affected vertices. We start with the following definition of *loop cover* of a vertex, which we use to speed up our backward search.

Definition 1. *Let $w \in D'(r'_y) \setminus S$ be a canonical vertex, and let ℓ_{min} be the ancestor of w in L with the lowest level such that $\ell_{min} \in D'(r'_y)$. Moreover, let (p, q) be the bridge such that $p \in D'_{\ell_{min}}$ and q is an ancestor of w. We call q the* loop cover *$lcover(w) = q$ of w in D. If $\ell(w) \notin D'(r'_y)$, then $lcover(w) = \emptyset$.*

We use the loop cover of vertices that are neither D-scanned nor L-affected in order to avoid unnecessary visits to vertices during the search for L-affected vertices. More specifically, whenever we visit a vertex $w \in D'(r'_y) \setminus D'_y$ that is not L-affected, then we do not need to visit any of the vertices in $D'(lcover(w))$. Formally, we have the following lemma.

Lemma 5. *Let $w \in D'(r'_y) \setminus D'_y$ be a canonical vertex that is not D-scanned and has a path to x in $G'[D'(r'_y)]$ and $\ell'(w') = c'_{\ell(w)}$. If $lcover(w) \neq \emptyset$, for every canonical vertex $v \in D'(lcover(w))$ such that v has a path to w in $G'[D'(lcover(w))]$, we have that $\ell'(v') = c'_{\ell(v)}$. If $lcover(w) = \emptyset$, for all vertices $v \in D'(r'_y)$ that have a path to w in $G'[D'(r'_y)]$, it holds that $\ell'(v') = c'_{\ell(v)}$.*

Lemma 5 allows us to traverse only the edges of L-affected vertices, and spend $O(1)$ time for each vertex that is not L-affected. As a result, assuming that a bridge is not locally canceled by the edge insertion, the set S' of L-affected vertices can be identified and L' can be correctly updated in time $O(V(S') + E(S') + n)$. Let v be a canonical vertex, and let t be the number of bridges on all paths from s to v. Then, we can show that $\ell'(v')$ can change at most $t \leq n - 1$ times. Thus, we obtain the following result.

Theorem 1. *Let G_s be a flow graph with n vertices. We can maintain the hyper-loop nesting forest L of G_s through a sequence of edge insertions in $O(mn)$ total time, where m is the number of edges after all insertions.*

4 Strong Connectivity Queries and 2-Vertex-Connected Components Under Edge Insertions

The data structure from [12] computes the strong bridges of G plus four trees: the dominator tree D and the loop nesting tree H of the flow graph G_s, and the dominator tree D^R and the loop nesting tree H^R of the reverse flow graph G^R_s. This information is sufficient to answer in optimal time all the queries considered in this paper. In particular, the crux of the method is the following theorem, which shows that the information relevant for our queries can indeed be extracted from the strong bridge of G and the four trees D, D^R, H and H^R:

Theorem 2 **([12]).** *Let $G = (V, E)$ be a strongly connected digraph, s be an arbitrary start vertex in G, and let $e = (u, v)$ be a strong bridge of G. Let C be a SCC of $G \setminus e$. Then one of the following cases holds:*

(a) If e is a bridge in G_s but not in G^R_s then either $C \subseteq D(v)$ or $C = V \setminus D(v)$.
(b) If e is a bridge in G^R_s but not in G_s then either $C \subseteq D^R(u)$ or $C = V \setminus D^R(u)$.
(c) If e is a common bridge of G_s and G^R_s then either $C \subseteq D(v) \setminus D^R(u)$, or $C \subseteq D^R(u) \setminus D(v)$, or $C \subseteq D(v) \cap D^R(u)$, or $C = V \setminus \big(D(v) \cup D^R(u)\big)$.

Moreover, if $C \subseteq D(v)$ (resp., $C \subseteq D^R(u)$) then $C = H(w)$ (resp., $C = H^R(w)$) where w is a vertex in $D(v)$ (resp., $D^R(u)$) such that $h(w) \notin D(v)$ (resp., $h^R(w) \notin D^R(u)$).

In this section, we show that exactly the same information can be extracted if we replace the loop nesting trees H and H^R with two new trees $\hat{H}$ and $\hat{H}^R$, which (differently from loop nesting trees) can be maintained efficiently throughout any sequence of edge insertions. As a result, the strong bridges of G plus D, D^R, $\hat{H}$ and $\hat{H}^R$ will allows us to answer all our queries in optimal time throughout any sequence of edge insertions.

We next define the new trees $\hat{H}$ and $\hat{H}^R$. Without loss of generality, we restrict our attention to $\hat{H}$, as $\hat{H}^R$ is defined analogously in G^R. We construct $\hat{H}$ starting from L, as follows. For every vertex u such that $c_u \neq u$ we set $\hat{h}(u) = c_u$, and for every vertex u where $c_u = u, u \neq s$ we set $\hat{h}(u) = \ell(u)$. Note that, once L is available, the tree $\hat{H}$ can be computed in $O(n)$ time. We can show that Theorem 2 still holds if we replace H by $\hat{H}$ and H^R by $\hat{H}^R$. Thus, in summary, our algorithm works as follows. Given a strongly connected digraph G subject to edge insertions, we maintain in a total of $O(mn)$ time the strong bridges of G [11], the dominator trees D and D^R [10], and the hyperloop nesting trees L and L^R, as shown by Theorem 1. After each edge insertion, we construct in $O(n)$ time the trees $\hat{H}$ and $\hat{H}^R$ from L and L^R, respectively. Since there can be at most m edge insertions, we spend $O(mn)$ time it total. By Theorem 2, after each update we can answer all our queries in optimal time.

In addition, we can answer all of our queries under vertex failures. Furthermore, we extend our algorithms to general (not necessarily strongly connected) digraphs: in this case, we maintain the hyperloop nesting tree in each SCC, as shown in the following theorem.

Theorem 3. *Let G be a general graph with n vertices. Both the dominator trees D and D^R, the hyperloop nesting trees L and L^R of each SCC C of G, all rooted at the same arbitrary start vertex s, can be maintained in total $O(mn)$ time under any sequence of edge insertions, where m is the number of edges after all insertions.*

All the details of the method will appear in the full version of the paper. The following theorem summarizes our results.

Theorem 4. *We can maintain a digraph G through any sequence of edge insertions in a total of $O(mn)$ time, where m is the number of edges after all insertions, and answer the following queries in asymptotically optimal (worst-case) time after each insertion:*

(i) Report in $O(1)$ time the total number of SCCs in $G \setminus e$ (resp., in $G \setminus v$), for a query edge e (resp., vertex v).
(ii) Report in $O(1)$ time the size of the largest and of the smallest SCCs in $G \setminus e$ (resp., in $G \setminus v$), for a query edge e (resp., vertex v).
(iii) Report in $O(n)$ time all the SCCs of $G \setminus e$ (resp., in $G \setminus v$), for a query edge e (resp., vertex v).
(iv) Test in $O(1)$ time if two query vertices u and w are strongly connected in $G \setminus e$ (resp., $G \setminus v$), for a query edge e (resp., vertex v).

(v) For query vertices u and w that are strongly connected in G, report all edges e (resp., vertices v) such that u and w are not strongly connected in $G \setminus e$ (resp., $G \setminus v$), in time $O(k)$, where k is the number of separating edges (resp., vertices).

Maintaining the 2-vertex-connected components under edge insertions. Using the incremental algorithm for maintaining the hyperloop nesting tree we are able to devise an incremental algorithm for maintaining the 2-vertex-connected components of a directed graph with n vertices in a total of $O(mn)$ time, where m is the number of edges after all insertions, and linear space. Our algorithm uses the high-level approach of the algorithm from [12], where the 2-vertex-connected components are computed in $O(n)$ time once the strong bridges, the dominator trees D and D^R, and the loop nesting trees H and H^R are given. The details of the algorithm are quite involved and different from Sect. 4, and are spelled out in the full version of the paper.

Theorem 5. *We can maintain the 2-vertex-connected components of a directed graph G through any sequence of edge insertions in a total of $O(mn)$ time, where m is the number of edges after all insertions, and linear space.*

5 Conditional Lower Bounds

In this section we present conditional lower bounds which imply that a polynomial improvement over our bounds would have interesting consequences, as such an improvement would disprove widely believed conjectures.

Conditional lower bound for the total update time in the partially dynamic version. We show that there is no partially dynamic algorithm that maintains a data structure that can answer the queries "are u and v strongly connected in $G \setminus e$?", where $u, v \in V$, $e \in E$, and has total update time $O((mn)^{1-\epsilon})$ (for some constant $\epsilon > 0$) and sub-polynomial query time unless the OMv Conjecture [14] fails. Under this conditional lower bound, the running time of our algorithm is asymptotically optimal up to sub-polynomial factors.

Theorem 6. *For any constant $\delta \in (0, 1/2]$ and any n and $m = \Theta(n^{1/(1-\delta)})$, there is no algorithm for maintaining a data structure under edge deletions or edge insertions allowing queries of the form "are u and v strongly connected in $G \setminus e$", where $u, v \in V$, $e \in E$, that uses polynomial preprocessing time, total update time $u(m, n) = (mn)^{1-\epsilon}$ and query time $q(m) = m^{\delta-\epsilon}$ for some constant $\epsilon > 0$, unless the OMv conjecture fails.*

Conditional lower bounds for the amortized/worst-case update time in the fully/partially dynamic setting. We prove conditional lower bounds for the problem of maintaining a data structure that can answer any query of Theorem 4. We base our bounds on the *Strong Exponential Time Hypothesis (SETH)* that was first stated in [15,16]. More specifically, under the SETH, we show that the trivial algorithm that recomputes the solution from scratch, using the static algorithm from [12], is asymptotically optimal up to sub-polynomial factors.

Theorem 7. *Let G be a digraph with n vertices that undergoes m edge updates from an initially empty graph (until the graph is empty in the decremental case). If for some $\epsilon > 0$, there exists an algorithm that can answer the following queries:*

(i) Report in $O(m^{1-\epsilon})$ time the total number of SCCs in $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v) in G.
(ii) Report in $O(m^{1-\epsilon})$ time the size of the largest SCC in $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v) in G.
(iii) Report in $O(m^{1-\epsilon})$ time all the SCCs of $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v).

while maintaining G fully dynamically with $O(m^{1-\epsilon})$ amortized update time and amortized query time after polynomial time preprocessing, or partially dynamically with $O(m^{1-\epsilon})$ worst-case update time and worst-case query time after polynomial time preprocessing, then the SETH is false.

We prove similar results of the last two query types of Theorem 4, in graphs where the number of edges is superlinear to the number of vertices.

Theorem 8. *Let $\epsilon > 0$ and $\delta > \epsilon/(1-\epsilon)$. Let G be a digraph with n vertices that undergoes $m > n^{1+\delta}$ edge updates from an initially empty graph (until the graph is empty in the decremental case). If there exists an algorithm that can answer the following queries:*

(iv) Test in $O(m^{1-\epsilon}/n)$ time whether two query vertices u and v are strongly connected in $G \setminus e$ (resp., $G \setminus v$), for any query edge e (resp., vertex v).
(v) For any two query vertices u and v that are strongly connected in G, test whether there exists an edge e (resp., vertex v) such that u and v are not strongly connected in $G \setminus e$ (resp., $G \setminus v$) in time $O(m^{1-\epsilon}/n)$.

while maintaining G fully dynamically with $O(m^{1-\epsilon})$ amortized update time and amortized query time, or partially dynamically with $O(m^{1-\epsilon})$ worst-case update time and worst-case query time, after arbitrary polynomial time preprocessing, then the SETH is false.

References

1. Abboud, A., Vassilevska Williams, V.: Popular conjectures imply strong lower bounds for dynamic problems. In: FOCS, pp. 434–443 (2014)
2. Alstrup, S., Harel, D., Lauridsen, P.W., Thorup, M.: Dominators in linear time. SIAM J. Comput. **28**(6), 2117–2132 (1999)
3. Aspnes, J., Chang, K., Yampolskiy, A.: Inoculation strategies for victims of viruses and the sum-of-squares partition problem. J. Comput. Syst. Sci. **72**(6), 1077–1093 (2006)

4. Buchsbaum, A.L., Georgiadis, L., Kaplan, H., Rogers, A., Tarjan, R.E., Westbrook, J.R.: Linear-time algorithms for dominators and other path-evaluation problems. SIAM J. Comput. **38**(4), 1533–1573 (2008)
5. Cohen, R., Havlin, S., Ben-Avraham, D.: Efficient immunization strategies for computer networks and populations. Phys. Rev. Lett. **91**, 247901 (2003)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
7. Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Algorithms and Theory of Computation Handbook, 2nd edn, vol. 1, pp. 9:1–9:28. CRC Press (2009)
8. Franciosa, P.G., Gambosi, G., Nanni, U.: The incremental maintenance of a depth-first-search tree in directed acyclic graphs. Inf. Process. Lett. **61**(2), 113–120 (1997)
9. Georgiadis, L., Hansen, T.D., Italiano, G.F., Krinninger, S., Parotsidis, N.: Decremental data structures for connectivity and dominators in directed graphs. In: ICALP, pp. 42:1–42:15 (2017)
10. Georgiadis, L., Italiano, G.F., Laura, L., Santaroni, F.: An experimental study of dynamic dominators. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 491–502. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33090-2_43
11. Georgiadis, L., Italiano, G.F., Parotsidis, N.: Incremental 2-edge-connectivity in directed graphs. In: ICALP, pp. 49:1–49:15 (2016)
12. Georgiadis, L., Italiano, G.F., Parotsidis, N.: Strong connectivity in directed graphs under failures, with applications. In: SODA, pp. 1880–1899 (2017)
13. Gunawardena, J.: A linear framework for time-scale separation in nonlinear biochemical systems. PLoS ONE **7**(5), e36321 (2012)
14. Henzinger, M., Krinninger, S., Nanongkai, D., Saranurak, T.: Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In: STOC, pp. 21–30 (2015)
15. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001)
16. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
17. Italiano, G.F., Laura, L., Santaroni, F.: Finding strong bridges and strong articulation points in linear time. Theor. Comput. Sci. **447**, 74–84 (2012)
18. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: KDD, pp. 137–146 (2003)
19. Kuhlman, C.J., Anil Kumar, V.S., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J.: Finding critical nodes for inhibiting diffusion of complex contagions in social networks. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6322, pp. 111–127. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15883-4_8
20. Mihalák, M., Uznański, P., Yordanov, P.: Prime factorization of the Kirchhoff polynomial: compact enumeration of arborescences. In: ANALCO, pp. 93–105 (2016)
21. Paudel, N., Georgiadis, L., Italiano, G.F.: Computing critical nodes in directed graphs. In: ALENEX, pp. 43–57 (2017)
22. Ramalingam, G., Reps, T.: An incremental algorithm for maintaining the dominator tree of a reducible flowgraph. In: POPL, pp. 287–296 (1994)

23. Shen, Y., Nguyen, N.P., Xuan, Y., Thai, M.T.: On the discovery of critical links and nodes for assessing network vulnerability. IEEE/ACM Trans. Netw. **21**(3), 963–973 (2013)
24. Tarjan, R.E.: Edge-disjoint spanning trees and depth-first search. Acta Informatica **6**(2), 171–85 (1976)
25. Ventresca, M., Aleman, D.: Efficiently identifying critical nodes in large complex networks. Comput. Soc. Netw. **2**(1), 1–16 (2015)

Efficient Algorithms for Listing k Disjoint st-Paths in Graphs

Roberto Grossi(✉), Andrea Marino(✉), and Luca Versari(✉)

Università di Pisa, Pisa, Italy
{grossi,marino,versari}@di.unipi.it

Abstract. Given a connected graph G of m edges and n vertices, we consider the basic problem of listing all the choices of k vertex-disjoint st-paths, for any two input vertices s, t of G and a positive integer k. Our algorithm takes $O(m)$ time per solution, using $O(m)$ space and requiring $O(F_k(G))$ setup time, where $F_k(G) = O(m \min\{k, n^{2/3} \log n, \sqrt{m} \log n\})$ is the cost of running a max-flow algorithm on G to compute a flow of size k. The proposed techniques are simple and apply to other related listing problems discussed in the paper.

1 Introduction

Listing paths of various kinds is a classical problem in graphs [7,13,17,18], as it models problems in several contexts. The survey in [4], for instance, provides several bibliographical references to applications in biological sequence alignment, natural language processing, speech recognition, reconstruction of metabolic path-ways, gene regulation networks, motion tracking, message routing in communications networks, power line placement, vehicle and transportation routing, building evacuation planning, timing analysis of circuits, task scheduling, VLSI layout, communications and transportation network design.

In this paper we consider the basic problem of listing all choices of k disjoint st-paths for a given connected graph G of m edges and n vertices, where s, t are two vertices of G and k is a positive integer. In other words, we want to list all the possible ways of connecting s and t using k vertex-disjoint paths, for both directed and undirected graphs. We propose an algorithm that takes $O(m)$ time per listed solution, using $O(m)$ space and requiring $O(F_k(G))$ setup time, where $F_k(G) = O(m \min\{k, n^{2/3} \log n, \sqrt{m} \log n\})$ is the cost of running a max-flow algorithm on G to compute a flow of size k.

Our algorithm can be seen as an example where the textbook algorithms for computing strongly connected components (SCCs) and maximum flows are applied in a simple way. For this, we reduce the problem on k vertex-disjoint paths to the one on k *edge*-disjoint *trails*. A trail is a walk, possibly containing cycles, but with no repeated arc (see Sect. 2). We list the k edge-disjoint trails from s to t, considering just the trails for which there exist other (recursively checkable) $k-1$ edge-disjoint trails reaching t. The existence of the latter trails is guaranteed by the existence of a flow of size k. We exploit the fact that

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 544–557, 2018.
https://doi.org/10.1007/978-3-319-77404-6_40

Table 1. Cost per solution when listing the k vertex-disjoint (unbounded or bounded) paths originating from any given vertex s, whether the target t is given or not.

	Unbounded length		Bounded length	
	Target given	Target not given	Target given	Target not given
$k = 1$	[7] and this paper (directed): $O(m)$ [2] (undirected): optimal		[14]: $O(nm)$ (directed) and $O(m)$ (undirected) **This paper**: $O(m)$	
$k = 2$	**This paper**: $O(m)$	[1] and this paper: $O(m)$	[11]: hard	[15]: $O(nm)$ **This paper**: $O(m)$
$k \geq 3$	**This paper**: $O(m)$		[11]: hard	[15]: hard

recomputing the flow for the next set of edge-disjoint trails takes $O(m)$ time instead of $O(F_k(G))$.

It is worth noting that some previous results, listed below, can be seen as variations of our problem. Our algorithm provides solutions within the same bounds as those reported in Table 1, and later discussed in Sect. 3, for the following ones:

– st-paths of a directed or undirected graph [7],
– shortest st-paths [8], a special case of bounded length st-paths [14],
– bubbles, i.e. pairs of vertex-disjoint directed st-paths, of fixed length [1],
– bubbles starting from a given vertex s [15],
– k-disjoint shortest paths with k given pairs of sources and targets [3].

Looking at the literature we also observe that the problem of finding k disjoint paths has been intensively studied. For $k = 1$, it is the classical problem of listing paths [2,7,13,17]. For $k > 1$, it is related to graph connectivity (e.g. Menger's theorem [12] and its extensions) and the papers [1,3,8,14,15] in the above list present some variations. When k ordered pairs (s_i, t_i) of vertices are specified, and s_it_i-paths are sought for, the problem of finding these k disjoint paths is NP-hard for arbitrary k in undirected graph [9], and even for $k = 2$ in directed graphs [5]. When k is fixed, an $O(n^3)$-time algorithm for undirected graphs can be obtained as a byproduct of the Robertson-Seymour papers on graph minors, and the bound has been recently improved to $O(n^2)$ time [10]. Note that the problem becomes polynomial when the k disjoint paths start from a source set $S = \{s_1, \ldots, s_k\}$ and reach a destination set $T = \{t_1, \ldots, t_k\}$, as the k disjoint paths can mix their sources and destinations (e.g. an s_it_j-path with $j \neq i$ is allowed).[1] Still, listing the latter paths is interesting in applications.

Preliminaries. Given a directed graph $G = (V, E)$, we refer to its number of vertices as n and to its number of arcs as m. We refer to a walk as a sequence of adjacent vertices and arcs. In the following we distinguish between *trails* and *paths*: the former ones correspond to walks where the arcs are all distinct, while the latter ones are loopless trails, i.e. walks where both arcs and vertices are all distinct. Given $s, t \in V$, a st-trail, also denoted as $s \leadsto t$, (respectively st-path) is a trail (respectively a path) which starts in s and ends in t. Let τ be a $s \leadsto u$ trail, we denote as $G \setminus \tau$ as the graph G without the arcs in τ. For an arc (u, v), we denote as $s \leadsto u \cdot (u, v)$ the extension of τ with the new arc.

[1] Connect a dummy source s to each s_i, and each t_i to a dummy target t; then, run a max flow algorithm from s to t.

2 Edge-Disjoint Trails to a Single Target in Directed Graphs

Given a directed graph $G = (V, E)$ and some vertices $s_1, \ldots, s_k, t$, we first describe how to list all the sets of k edge-disjoint trails $\tau_1, \ldots, \tau_k$ such that τ_i is a $s_i t$-trail. After that, for vertices s, t, we will describe how to list edge-disjoint trails $\tau_1, \ldots, \tau_k$ such that τ_i is a st-trail.

Let us first focus on the well-known case $k = 1$. To generate all the $s_1 t$-trails, we can adopt a recursive algorithm that starts out with $u = s_1$ and considers the current trail $s_1 \rightsquigarrow u$ (seen as a sequence of arcs or a set of arcs with a little abuse of notation). It then explores in a recursive fashion, one at a time, each good neighbor v of u: namely, v is good iff v can reach t in the reduced graph $G' \equiv G \setminus (s_1 \rightsquigarrow u)$. For each good neighbor v, the recursion proceeds with the extended trail $s_1 \rightsquigarrow v \equiv (s_1 \rightsquigarrow u) \cdot (u, v)$.

For $k > 1$, we observe that not all choices of τ_1 are fruitful, as some of them could lead to dead ends for the remaining $k - 1$ trails. During the generation of each trail τ_1, we say that v is a *good neighbor* of u iff v reaches t in the reduced graph $G' \equiv G \setminus (s_1 \rightsquigarrow u)$, *and* there are also $k - 1$ disjoint trails from $s_2, \ldots, s_k$ to t: these two conditions can be equivalently seen as recursively checking the existence of k disjoint trails from $v, s_2, \ldots, s_k$ to t in G'.

We can therefore see the above generation of k disjoint trails as a recursive scheme, where the current $s_1 u$-trail $(s_1 \rightsquigarrow u)$ has been already explored (initially, $u = s_1$). The main two cases are handled as follows.

- If $u = t$, the currently found trail $s_1 \rightsquigarrow u$ is τ_1. Moreover, if $k > 1$, recursively proceed with the $k - 1$ disjoint trails from vertices $s_2, \ldots, s_k$, setting $u := s_2$ and $G := G \setminus \tau_1$ (and noting that all these solutions will have this τ_1 fixed).
- If $u \neq t$, continue to generate the feasible trails τ_1 extending $s_1 \rightsquigarrow u$ (as prefix) in G. For this, extend u with its good neighbors: for each good neighbor v of u, proceed recursively with the extended trail $s_1 \rightsquigarrow v \equiv (s_1 \rightsquigarrow u) \cdot (u, v)$, traversing arc (u, v) and setting $u := v$.

The resulting recursion tree is illustrated in Fig. 1(a). It is made up by a top tree for all the feasible choices of τ_1, where its leaves on the first dashed level are in one-to-one correspondence with these choices. Each such leaf is the root of the recursion tree for τ_2, where the leaves on the second dashed level are in one-to-one correspondence with these choices. Each such leaf is the root of the recursion tree for τ_3, and so on. In the specific case of Fig. 1(a), nodes a, b, c, d, e indicate the possible choices for τ_1, while nodes f, g, h (respectively j, l, m) indicate the possible choices for τ_2 when $\tau_1 = b$ (respectively $\tau_1 = e$). Finally, nodes u, v, x, y correspond to k-sets of disjoint trails $\tau_1, \ldots, \tau_k$ where τ_i is the trail traversed at the i-th dashed line in the trail towards r in this tree.

In the following, we will call *nodes* the ones in the recursion tree to distinguish them from the *vertices* in the input graph.

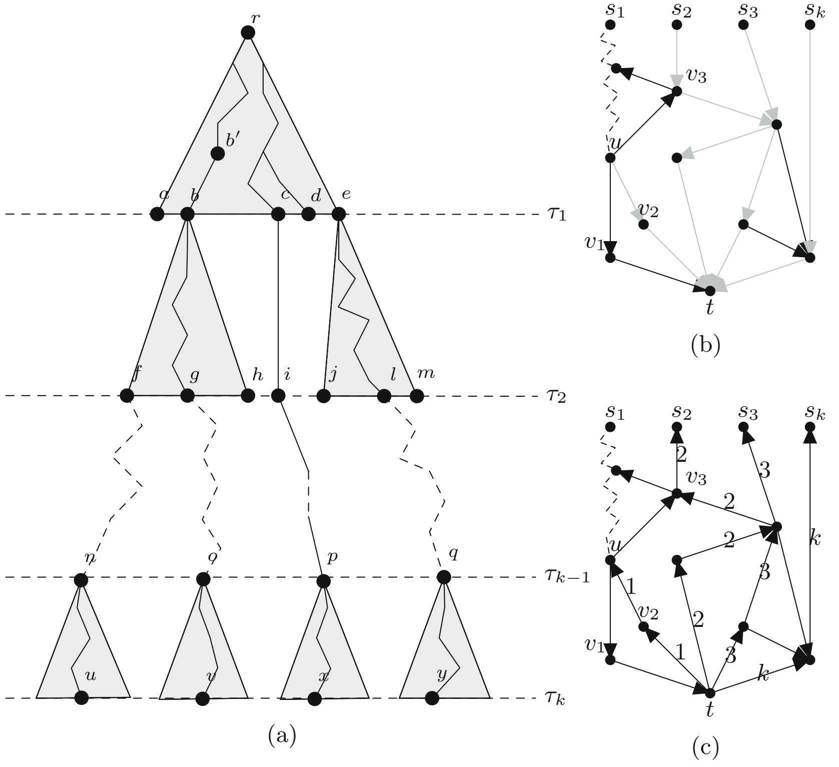


Fig. 1. (a) Recursion tree. (b) The paths $\pi_1, \ldots, \pi_k$ in gray with the dashed trail $s_1 \rightsquigarrow u$ and (c) the corresponding certificate C.

Introducing the certificate. In order to make this recursion efficient within our claimed bounds, we introduce a certificate for each node in the recursion tree, remarking the difference between trails and paths. We observe that the partial trail $s_1 \rightsquigarrow u$ uniquely identifies a node in the recursion tree where branching occurs by considering vertex u and its good neighbors. We want to keep track of a choice of k disjoint trails $\tau_1, \tau_2, \ldots, \tau_k$ from $s_1, \ldots, s_k$ to t, so that $s_1 \rightsquigarrow u$ is a prefix of τ_1. We observe that there exist k disjoint trails from $u, s_2, \ldots, s_k$ to t iff there exist k disjoint *paths* from $u, s_2, \ldots, s_k$ to t. Hence let $\pi_1, \pi_2, \ldots, \pi_k$ be the latter paths, and $G' \equiv G \setminus (s_1 \rightsquigarrow u)$ be the reduced graph, with the arcs in the partial trail removed.

Definition 1. *The* certificate *is an augmented graph $C = (V_C, E_C)$ defined in terms of the partial trail $s_1 \rightsquigarrow u$ and the paths $\pi_1, \pi_2, \ldots, \pi_k$, where the arcs belonging to the latter paths are reversed*[2] *and labeled to keep track of their membership to the paths, namely,*

[2] Note that the certificate can be seen as the residual network in max-flow on arcs with $0-1$ capacities.

– *$V_C = V$ is equal to the vertex set of G, and*
– *$E_C = \{(x, y) : (y, x) \in E(G') \cap \Pi(G)$ or $(x, y) \in E(G') \setminus \Pi(G)\}$, where $G' \equiv G \setminus (s_1 \rightsquigarrow u)$ and $\Pi(G) = \cup_{i=1}^{k} \pi_i$ denotes the set of arcs belonging to the paths.*
– *Each arc (x, y) in E_C such that $(y, x) \in \pi_i$ ($1 \le i \le k$) is endowed with the label i.*

An example of certificate for the paths $\pi_1, \ldots, \pi_k$ and the trail $s \rightsquigarrow u$ in Fig. 1(b) is shown in Fig. 1(c), where each arc with label i belongs to a path π_i from s_i to t reversed.

Lemma 1. *Given a trail $s_1 \rightsquigarrow u$, the certificate C can be built in $O(F_k(G))$ time.*

Proof. Given $s_1, \ldots, s_k$ and t, we modify G by connecting $s_1, \ldots, s_k$ to a dummy vertex s. We can apply the Ford Fulkerson algorithm to get in $O(km)$ time k disjoint paths from s, and hence from $s_1, \ldots, s_k$, to t. In order to use a faster max-flow algorithm, like [6], we can do the following. Run the max-flow algorithm to find all the edges E' used by a flow from s to t of size k, then add k arcs from t to s to E'. In order to build our certificate, we need to find out k disjoint paths using the edges in E'. To this aim, we notice that all the vertices in the graph induced by the edges in E' have the in-degree equal to their out-degree. As this graph is an Eulerian multidigraph, we can find an Eulerian cycle involving s which by deleting the k arcs (t, s) splits into k disjoint st-trails. By deleting possible cycles in these trails, we get k disjoint st-paths. □

As discussed next, we employ the certificate to

– guarantee that there is at least one good neighbor of u, i.e., no dead ends for the current node in the recursion tree;
– locate the next good neighbor of u and save space (by avoiding to keep lists of good neighbors at each node in the recursion stack);
– quickly skip unary nodes in the recursion tree, i.e. when u has just a single good neighbor.

In this way the recursion tree in Fig. 1(a) is actually flattened as a single tree where each node has at least two children. The benefit is clear by allocating a budget of $O(m)$ time on each child for every node, since we obtain an $O(m)$ cost per listed solution as a result. On the downside, since C extends the partial trail $s_1 \rightsquigarrow u$ to a specific choice of k disjoint trails, we have to characterize how to explore the other choices and, for each such choice, how to update C without computing it from scratch each time (hence, with a lower cost than that stated in Lemma 1).

At least one good neighbor of u. This part follows immediately from the definition of certificate, as the vertex v following u in the trail τ_1 can be characterized as the only vertex in G' that is neighbor of u and has a reverse arc in C. For this, we call v the *favorite* neighbor of u.

Next good neighbor of u. Apart from the favorite neighbor of u, mentioned above, we want to identify all the remaining good neighbors v of u. Also, the identification of the good neighbors should rely on a property that holds for any choice of the certificate C, so that the underlying C does not really matter to perform this task correctly, as shown below.

Lemma 2. *Given trail $s_1 \rightsquigarrow u$, for any certificate C and for any arc (u, v), we have that v is a good neighbor of u iff either v is the favorite neighbor of u in C or there is a cycle in C going through (u, v).*

Proof. If v is the favorite neighbor of u in C, we have nothing to prove. Otherwise, we want to prove that there is another certificate C' in which v is the favorite neighbor of u if and only if (u, v) is involved in a cycle in C. Consider the flow f that corresponds to certificate C and suppose that there is another flow d' on the same network that goes through (u, v). Then, it is well known that there will be a cycle in the residual network (i.e. in C) that contains the symmetric difference of the edges in the two flows. Moreover, the converse also holds. So we have another flow (and so another certificate) in which v follows u if and only if there is a cycle in C that uses (u, v). $\square$

As an example, consider Fig. 1(c). The good neighbors of u are v_2 (favorite neighbor) and v_1 (since it creates a cycle in C). On the other hand, v_3 is not a good neighbor for u as it does not satisfy both these conditions.

An immediate application of Lemma 2 to the nodes in the recursion tree has an excessive cost in terms of space and time.

Let us start with space. Rather than storing the list of good neighbors at each node in the recursive stack, which may take $\Omega(m)$ space since the same vertex can appear several times in the partial trail $s_1 \rightsquigarrow u$, we want to find the next good neighbor v' (if any) starting from the knowledge that the current good neighbor is v. Since vertices are numbered, it suffices to produce the list of good neighbors on the fly each time, and then select v' as the smallest one that is greater than v. This costs $O(m)$ time and fits the $O(m)$-per-children budget allocated to each node in the recursion tree.

Lemma 3. *Given trail $s_1 \rightsquigarrow u$ and any certificate C for it, the sorted list of good neighbors can be returned in $O(m)$ time and space.*

Proof. By Lemma 2, one neighbor is the favorite one that can be retrieved from C. As for the remaining ones, we compute the strongly connected components (SCCs) of C. Now, each arc (u, v) that is in a cycle must belong to a SCC that contains u. Thus scanning the SCCs identifies these arcs, as required by Lemma 2. Radix sorting these neighbors gives the wanted list. Total cost is $O(m)$ time and space. $\square$

As for the time, rather than building the certificate in each node of the recursion tree as stated in Lemma 1, we prefer to compute the certificate from the parent or a child, taking $O(m)$ time instead of $O(F_k(G))$. In other words, we run the max-flow algorithms mentioned in Lemma 1 only once, to find one certificate at the beginning of the recursion. After that, all the other certificates are computed using the method based on the following lemma.

Lemma 4. *Given the current node of the recursion tree and any certificate C for its trail $s_1 \rightsquigarrow u$, the certificate for the parent or for a child of the node can be computed in $O(m)$ time and space.*

Proof. Let $s_1 \rightsquigarrow u'$ be the trail in the parent node, where u is a good neighbor of u' and so $s_1 \rightsquigarrow u = (s_1 \rightsquigarrow u') \cdot (u', u)$. The new certificate is obtained from C by just adding the reversed arc (u, u') to C with label 1. Indeed, note that (u', u) was deleted in C, as C refers to $G \setminus s_1 \rightsquigarrow u$, and in the new certificate we are setting $\pi_1 := (u', u) \cdot \pi_1$.

We now show how the certificate modifies while going into a child node. Let $s_1 \rightsquigarrow v = (s_1 \rightsquigarrow u) \cdot (u, v)$ be the trail in the child node, where v is a good neighbor of u. Adding (u, v) may kill at most two paths in C: the path π_1 as we are possibly choosing another path replacing it, and a path π_i with $i \neq 1$, which could traverse (u, v). (Note that π_i is killed as we want disjoint paths, and no other π_j with $j \neq 1, i$ can traverse (u, v) for the same reason.) Observing that C can be seen as a residual network, we use the property of augmenting paths in Ford-Fulkerson algorithm. In particular, we have $k - 2$ flows (using a dummy source connected to $v, s_2, \ldots, s_k$), so we can run twice the $O(m)$-time algorithm to find an augmenting path to bring back the flow to k. This immediately translates into finding k disjoint trails.

Note that a final check is required to see if we still have paths (loop less trails) in the reversed arcs of the resulting certificate: in that case, we just remove the loops from the trails in $O(m)$ time. □

No unary nodes in the recursion tree. We need to avoid unary (i.e. single-child) nodes in the recursion tree. We have two kinds of situations to deal with as illustrated in Fig. 1(a). A unary node can occur when producing a trail τ_i (as there is only one good neighbor), or when switching on a leaf from τ_i to the root that will generate τ_{i+1} (when for the given choice of $\tau_1, \ldots, \tau_i$, there is only one trail τ_{i+1} that exist). Looking at Fig. 1(a), the former case corresponds to unary paths which are internal to a tree, as for instance the one from b' to b, while the second case corresponds to chains which remain unary across the trees, as for instance the one from c to p. In particular, this latter situation can give rise to a dependency on k in the time complexity ($O(mk)$) that suitably disappears by avoiding unary nodes.

We handle both kinds of situations using a simple "fast forward" technique that takes $O(m)$ time to skip maximal paths of unary nodes in the recursion tree, so that we actually obtain a compact recursion tree where each node always has two or more children.

Lemma 5. *Given the current node of the recursion tree and any certificate C for its trail $s_1 \rightsquigarrow u$, we can skip a unary chain starting from u in the recursion tree in $O(m)$ time and space.*

Proof. It is an extension of what discussed in the proof of Lemma 3. We compute the SCCs in the certificate C as before. We observe that if u is unary then the SCC containing u is trivial. Since the SCCs form a DAG, we take the sequence $S_1, \ldots, S_l$ of SCCs that are traversed by π_1 (see Definition 1 for the definition of π_i), where S_1 is the trivial SCC containing u and S_l is the SCC containing t. We take the first vertex u' along π_1 (towards t) that follows u and belongs to a non-trivial SCC S_j. Now we know that there are at least two good neighbors for u', which can be found in S_j as already discussed in the proof of Lemma 3. This handles the former situation.

To handle the latter situation, we observe that when we traverse $S_1, \ldots, S_l$ we always find trivial SCCs. So we switch to π_2, which we know from C, and take the sequence $S'_1, \ldots, S'_l$ of SCCs that are traversed by π_2. We stop in the first $S'_{j'}$ that is non-trivial. If it does not exist, we switch to π_3, and iterate the same approach. Either we stop at a non-trivial SCCs, and we proceed as in the proof of Lemma 3, or we find all trivial SCCs till π_k, so we obtain a solution to list.

In both situations, the cumulative cost is $O(m)$ time and space as we always traverse distinct SCCs. □

Theorem 1. *Given a directed graph G and its distinct vertices $s_1, \ldots, s_k, t$, all the k-sets of edge-disjoint trails $\tau_1, \ldots, \tau_k$, such that τ_i is a $s_i t$-trail for $1 \le i \le k$, can be listed with $O(m)$ time cost per solution. Initial setup time is $O(F_k(G))$ and space usage is $O(m)$.*

Proof. By definition of good neighbors, given a partial trail $s_i \rightsquigarrow u$ and the already generated trails $\tau_1, \ldots, \tau_{i-1}$, a simple induction shows that we generate all the k disjoint trails having $\tau_1, \ldots, \tau_{i-1}$ fixed and prefix $s_i \rightsquigarrow u$ for the ith trail. There is no solution listed twice for the same reason. Also, suppose by contradiction that there exists a solution that is not generated. Let $\tau_1, \ldots, \tau_{i-1}$ fixed and prefix $s_i \rightsquigarrow u$ be the node in the recursion tree that fails to generate it. By definition of good neighbor, this is a contradiction, as we identify the next v to be taken, so that $s_i \rightsquigarrow u$ can be extended to $s_i \rightsquigarrow u \cdot (u, v)$ instead, where v is the next vertex on the missed trail. Hence, all the trails are correctly listed once. Since we spend $O(m)$ time in each node of the recursion tree, and there are no unary nodes, we get $O(m)$ time per solution. The space is $O(m)$ as a trail can be so long, and we use constant memory per node in the recursion stack, which is of depth $O(m)$. □

We can now deal with the case where we have a single source s.

Theorem 2. *Given a directed graph G and two vertices $s, t \in V$, with $s \neq t$, all the k-sets of edge-disjoint trails $\tau_1, \ldots, \tau_k$, such that τ_i is a st-trail for $1 \le i \le k$, can be listed with $O(m)$ time cost per solution, setup time $O(F_k(G))$, and space usage $O(m)$.*

Proof. Let the *k-starting sets* be the k-subsets of s's neighbors, i.e. $\{s_1, \ldots, s_k\} \subseteq N(s)$, such that there are k disjoint paths $\pi_1, \ldots, \pi_k$ (and hence trails) where

π_i is an $s_i t$ path. If we apply Theorem 1 to every k-starting set, we obtain our claim.

Since we cannot explore all the possible subsets of $N(s)$, we again use a recursive approach to generate just the k-starting sets of s. Suppose that $N(s)$ is sorted by vertex numbering. At each recursive step we divide the vertices of $N(s)$ into three groups:

- I = those vertices that will be part of any starting set generated from the current recursive call;
- H = those vertices that won't be in any starting set generated from the current recursive call;
- U = those vertices that may or may not be in a starting set.

Initially, both I and H are empty, and $U = N(s)$.

During the recursion, we keep the invariant that U forms a contiguous suffix of the sorted sequence $N(s)$. At each step, we compute a flow f of cardinality k starting from $\{s\} \cup I$ on the graph obtained from G by erasing all incoming edges in s and all outgoing edges from s to any vertex in $I \cup H$. More precisely, we require one unit of flow to leave each vertex in I, none to leave any vertex in H, and $k - |I|$ to leave s.[3] We observe that we are actually enforcing a flow of cardinality k that uses first all the vertices in I. This flow is computed from scratch once, at the beginning of the recursion. In the rest of the recursion, it is updated through the calls using the same ideas as in the proof of Lemma 4.

We compute the residual network R_f of f, and the SCCs of R_f. We can thus build the set W of vertices $v \in U$ such that (s, v) is involved in a cycle in R_f.

If W is empty, then there is exactly one k-starting set that satisfies the constraints given by the sets I and H, so we can output I (plus any other neighbor of s involved in the f) as k-starting set, and return from the recursive call.

Otherwise, let v be the smallest vertex in W. Since (s, v) is involved in a cycle in R_f (as any other vertex in W), there must exists at least two flows, both traversing the vertices I and avoiding those in H, such that one flow traverses (s, v) and the other does not traverse (s, v). This means that the current recursive call generates two further calls: the former where v goes into I, and the latter v goes into H. In both calls, all vertices in U that precede v go into either I or H, according to their role in f: a vertex $x \in U$ with $x < v$, goes to I if x is traversed by the flow for the call at hand, or goes to H otherwise.

Hence, each calls returns at least one solution, and each internal call gives raise to two further calls. Hence the cost per solution is bounded by the cost of a single call, which is $O(m)$. Space cost is $O(m)$ with setup time $O(F_k(G))$.

It is worth observing that for each generated k-starting set, we can produce in $O(m)$ time the initial certificate needed (with the empty path as the current path). In this way we do not pay each time the setup cost $O(F_k(G))$ when applying Theorem 1 to the k-starting set. □

[3] This flow can be achieved by creating a dummy vertex s' connected to all the vertices in I with capacity 1 and to s with an arc of capacity $k - |I|$.

3 Applications to Related Problems

In this section, we show how to use the results in Sect. 2 to solve a variety of related problems. We firstly discuss some easy variations, including listing vertex disjoint paths, and then we focus on the problem of listing bounded-length fixed-source k-disjoint-paths with $k = 2$, since for $k \geq 3$ (as summarized in Table 1) the latter problem is hard.

3.1 Vertex-Disjoint Paths and Other Variations

Undirected graphs and vertex disjoint paths. Since we are interested in edge disjoint trails, in the case of undirected graphs, it is not sufficient to consider two opposite arcs in place of an undirected edge, as we want to avoid to traverse one arc in the solution if the other has been used. To this aim, well-known reductions [16] allow to easily extend our Theorem 2 to undirected graphs. Moreover, by transforming each vertex v into an arc (v_{in}, v_{out}), putting arcs (w_{out}, v_{in}) and (v_{out}, z_{in}) for each in-neighbor w and each out-neighbor z of v, vertex disjoint paths reduce to edge disjoint trails: as each of the k edge disjoint trails composing a solution do not use the same arc twice, we get the following.

Theorem 3. *Given a directed or undirected graph G and two vertices $s, t \in V$, with $s \neq t$, all the k-sets of vertex disjoint paths $\pi_1, \ldots, \pi_k$, such that π_i is a st-path for $1 \leq i \leq k$, can be listed with $O(m)$ time cost per solution, setup time $O(F_k(G))$, and space usage $O(m)$.*

It is worth observing, that Theorem 3 generalizes the result in [1] for any k, getting the same bounds for $k = 2$, as $O(F_2(G))$ is $O(m)$.

Cycles involving two vertices. Given an undirected graph, Theorem 3 easily extends to enumerating all the simple cycles that contain two given vertices s, t: it is enough to enumerate all the pairs of vertex-disjoint paths that connect s to t. As for directed graphs, we recall that finding even one directed cycle involving two vertices is NP-hard [5].

Multiple Sources vs Multiple Targets. Further variations can be considered. For instance, listing all the k-sets of edge disjoint trails (or vertex disjoint paths) from any subset of $x \geq k$ sources $\{s_1, \ldots, s_x\}$ to any subset of $y \geq k$ targets $\{t_1, \ldots, t_y\}$. This can be easily solved in the same bounds claimed by Theorems 2 and 3, by simply attaching $s_1, \ldots, s_x$ to a dummy source s and $t_1, \ldots, t_y$ to a dummy target t. We remark that this problem is different from the one considered by Robertson and Seymour and Kawarabayashi et al. [10] concerning the so called *disjoint paths problem*, since in their case, as said in the introduction, the sources and targets are paired and any algorithm is forced to find a k-set of vertex disjoint paths respecting this pairing.

Fixed source vs Variable Target. Let us now consider the following: given a directed graph $G = (V, E)$ and $s \in V$, list all the k-sets of edge disjoint st-trails (or vertex disjoint st-paths) for any $t \in V$. In order to solve this problem, we need to compute for which t there is at least a solution. By applying the results in Theorems 2 and 3, we get a total time cost of $O(nF_k(G) + \alpha m)$ with $O(m)$ space, where α is the number of solutions. In undirected graphs, for $k = 1, 2, 3$ the total time cost becomes simply $O(\alpha m)$ as the n flow computations can be replaced by the computation of k-connected components.

3.2 Bounded-Length Fixed-Source Two-Disjoint-Paths

In this section we discuss the following problem which deals with st-paths of bounded lengths.

Problem 1. Let $G = (V, E)$ be a directed graph, $s \in V$ and $\ell \in \mathbb{N}$. List all the k-sets of vertex disjoint st-paths of length at most ℓ, simultaneously for any $t \in V$.

State-of-art results are summarized in the last two columns of Table 1. If target t is given a priori along with source s, the only case which can be solved with output-sensitive bounds is for $k = 1$. Hence we focus on the case $k = 2$ with t not given a priori (Problem 1), showing how to apply similar ideas to Sect. 2. As this will make use of a subroutine to list all the st-paths of length at most ℓ, we will also focus on the case $k = 1$ where both s and t are given a priori.

Given s, we will make use of ball $B_\ell(G)$, which is defined as the graph induced by the vertices at distance at most ℓ from s in G. For the sake of simplicity, we assume wlog that s has zero indegree in G. Similarly to Sect. 2, we build first a st-path π_1 for some t and then the second vertex disjoint st-path π_2. Once the st-path π_1 is fixed, the suitable choices for π_2 are all the st-paths of length at most ℓ in G where vertices in π_1 except s and t have been deleted.

Let us now focus on the building process for π_1. Recall that while building π_1 we have to guarantee that π_1 can be completed in a solution, i.e. there is at least a suitable π_2 which can be paired with π_1. Let u be the current vertex explored during the recursion (at the beginning $u = s$) and let π'_1 be the su-path of length h with $h \leq \ell$ built until that point. As in the previous section we have to explore all the good neighbors of u. To discover them, we will employ an auxiliary graph G', obtained from G by removing all the vertices in π'_1 except s. A neighbor v of u is good whether there is a way to complete the partial solution, i.e. v is in G' and, moreover, in G' there is a vertex t such that there is a vt-path π''_1 with length at most $k - h$ and there is also a st-path π_2 of length at most ℓ, where π''_1 and π_2 are vertex disjoint. For each good neighbor v of u we recur by deleting v from G'. Note that this guarantees that we generate just simple paths and, by definition of good neighbors, the path π_2 does not overlap with π'_1 and π''_1.

Good Neighbors and Certificate. Similarly to the previous section, we maintain a certificate in order to recognize the good neighbors v for u. It may happen that u itself may be reached in at most ℓ steps from s without using any nodes used in π_1. In this case, we know that u is a valid target and so we proceed with generating all possible π_2. Moreover, it is also possible that π'_1 is a prefix of a valid path (this must be true if u is not a valid target, as we want to avoid dead-ends). In this case the certificate C is simply a ut-path from u to some vertex t in $B_\ell(G')$, such that t is the only vertex of the path in $B_\ell(G')$. Let v be the neighbor of u in C, where v corresponds to the *favorite* neighbor defined in Sect. 2. The other good neighbors are the vertices in G' that are in $N(u)$ and can reach at least a vertex in $B_\ell(G')$ using at most $\ell - h$ arcs. These can be easily computed in $O(m)$ by collapsing the vertices in $B_\ell(G')$ into a single vertex b and then running a backward BFS from b in G' truncated at distance $\ell - h$. Let R be the vertices reached by this BFS: the good neighbors for u are all the vertices in $N(u) \cap R$ and a certificate for each of them is their path to b. Hence, for any good neighbor, we have a child node in the recursion tree and for each of them we can build the certificate in $O(m)$. When backtracking from a child to its parent we can rebuild the certificate in the same way in $O(m)$ thus avoiding to store the past certificates in the recursion stack as in Sect. 2.

Fast Forward. We now show how to guarantee that in each recursion node we always have at least *two* children, we spend $O(m)$ time in each recursion node. We want to skip unary chains in the recursion tree in $O(m)$ time as in Lemma 5. Let u be the current vertex, let π'_1 be the su-path of length h with $h \leq \ell$ built until that point, and let π''_1 be the ut-path (for some t) in the current certificate C. There is a unary chain for instance if the neighbor v of u in π''_1 is its only good neighbor or if the path in π''_1 is the only feasible completion for π'_1, that is for each vertex in π''_1 except t, which is in $B_\ell(G')$, there is only one good neighbor. Precisely, we want to find the first vertex w in the path π''_1 that has at least two neighbors in $O(m)$ in order to skip all the intermediate vertices and continue the recursion from that one. This task can be easily addressed by modifying the certificate update as shown next.

1. Scan the vertices in the path π''_1 in their order checking whether there are at least two good neighbors as follows: let b be the vertex obtained by collapsing the vertices in $B_\ell(G')$, for each vertex w in the path π''_1 check whether there is an arc (w, z) not in π''_1 such that z is at distance at most $\ell - h$ from b.
2. Let w be the first vertex satisfying this condition. We directly add all the edges on the uw-path in π''_1 to π'_1 going directly to the recursion node corresponding to w.

With respect to Sect. 2, we are ignoring fast-forwarding across the recursion trees for the different values of k, as since $k = 2$, linear dependencies on k do not afflict here our asymptotic time costs per solutions.

Generating π_2. Once a st-path π_1 has been generated, we have to generate all the st-paths of length at most ℓ in the graph G' which is obtained by deleting from G all the vertices in π_1 except s and t. We show next that this can be done with a slight modification of the process for generating π_1 shown above. As in the above case, when recurring at u and enlarging the su-path π'_2 with length h, the certificate C is a path towards t of length at most $\ell - h$. Each recursive node explores all the good neighbors of u, i.e. the ones leading to t with at most $\ell - h$ arcs. The certificate update and the computation of the good neighbors can be done exactly as in the above case by simply replacing $B_\ell(G)$, and hence b, with the given target t. Hence, in order to understand which are the good neighbors, it suffices to do a backward BFS from t truncated at distance $\ell - h$. The good neighbors are the ones in $N(u)$ reached by this BFS and the certificate for them is their path to t. By replacing $B_\ell(G)$ with t in the π_1 generation, we get also fast-forward which allows to skip unary chains in the recursion tree in $O(m)$ time: as before, in $O(m)$ time we can scan the vertices w of the su-path in C looking for arcs (w, z) with z at distance at most $\ell - h$ from t.

Since both the trees generating π_1 and π_2 have no unary nodes and each internal node costs $O(m)$ time, we get a $O(m)$ cost per solution. Concerning space, we observe that certificates, as in Sect. 2, are updated not only while going from a parent node to a child, but also while backtracking. For this reason, the recursion stack for both π_1 and π_2 trees is just made by good neighbors lists whose cardinalities sum up to $O(m)$ since each vertex appears just once in each root to leaf path.

As a result, we get the following, which holds also for undirected graphs by simply replacing each edge with a pair of opposite arcs.

Theorem 4. *Let $G = (V, E)$ be a directed or undirected graph and $\ell \in \mathbb{N}$ and consider Problem 1.*

- *Given $s, t \in V$, there is an algorithm which lists all the st-paths of length at most ℓ with $O(m)$ time costs per solution (case $k = 1$).*
- *Given $s \in V$, there is an algorithm which lists all pairs of disjoint st-paths of length at most ℓ for any $t \in V$ with $O(m)$ time costs per solution (case $k = 2$).*

Setup time and space usage are $O(m)$ in both cases.

As a final remark, Theorem 4 also extends to the problem of listing all the cycles with bounded length involving a given vertex.

References

1. Birmelé, E., Crescenzi, P., Ferreira, R., Grossi, R., Lacroix, V., Marino, A., Pisanti, N., Sacomoto, G., Sagot, M.-F.: Efficient bubble enumeration in directed graphs. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 118–129. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34109-0_13
2. Birmelé, E., Ferreira, R.A., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., Sacomoto, G.: Optimal listing of cycles and st-paths in undirected graphs. In: Proceedings of the SODA, pp. 1884–1896 (2013)
3. Eilam-Tzoreff, T.: The disjoint shortest paths problem. Discrete Appl. Math. **85**(2), 113–138 (1998)

4. Eppstein, D.: k-best enumeration. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms, pp. 1003–1006. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-642-27848-8
5. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theor. Comput. Sci. **10**(2), 111–121 (1980)
6. Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. J. ACM (JACM) **45**(5), 783–797 (1998)
7. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. **4**(1), 77–84 (1975)
8. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. J. ACM **24**(1), 1–13 (1977)
9. Karp, R.M.: On the computational complexity of combinatorial problems. Networks **5**(1), 45–68 (1975)
10. Kawarabayashi, K., Kobayashi, Y., Reed, B.: The disjoint paths problem in quadratic time. J. Comb. Theory Ser. B **102**(2), 424–435 (2012)
11. Li, C.-L., McCormick, S.T., Simchi-Levi, D.: The complexity of finding two disjoint paths with min-max objective function. Discrete Appl. Math. **26**(1), 105–115 (1990)
12. Menger, K.: Zur allgemeinen kurventheorie. Fundam. Math. **10**(1), 96–115 (1927)
13. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks **5**(3), 237–252 (1975)
14. Rizzi, R., Sacomoto, G., Sagot, M.-F.: Efficiently listing bounded length st-paths. In: Jan, K., Miller, M., Froncek, D. (eds.) IWOCA 2014. LNCS, vol. 8986, pp. 318–329. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19315-1_28
15. Sacomoto, G., Lacroix, V., Sagot, M.-F.: A polynomial delay algorithm for the enumeration of bubbles with length constraints in directed graphs and its application to the detection of alternative splicing in RNA-seq data. In: Darling, A., Stoye, J. (eds.) WABI 2013. LNCS, vol. 8126, pp. 99–111. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40453-5_9
16. Schrijver, A.: Combinatorial optimization: polyhedra and efficiency. Springer Science & Business Media, Heidelberg (2003)
17. Tarjan, R.: Enumeration of the elementary circuits of a directed graph. SIAM J. Comput. **2**(3), 211–216 (1973)
18. Tiernan, J.C.: An efficient search algorithm to find the elementary circuits of a graph. Commun. ACM **13**, 722–726 (1970)

Transversals of Longest Cycles in Chordal and Bounded Tree-Width Graphs

Juan Gutiérrez(✉)

Departamento de Ciência da Computação, Universidade de São Paulo, Rua do Matão 1010, São Paulo, SP 05508–090, Brazil
juanguti@ime.usp.br

Abstract. Let $\mathrm{lct}(G)$ be the minimum size of a set of vertices that intersects every longest cycle of a 2-connected graph G. Let $\mathrm{tw}(G)$ be the tree-width of G and $\omega(G)$ be the size of a maximum clique in G. We show that $\mathrm{lct}(G) \leq \mathrm{tw}(G) - 1$ for every G, and that $\mathrm{lct}(G) \leq \max\{1, \omega(G) - 3\}$ if G is chordal. Those results imply as corollaries that all longest cycles intersect in 2-connected series-parallel graphs and in 3-trees. We also strengthen the latter result and show that all longest cycles intersect in 2-connected graphs of tree-width at most 3, also known as partial 3-trees.

1 Introduction

It is known that, in a 2-connected graph, every pair of longest cycles intersect each other in at least two vertices [1]. A natural question asks whether all longest cycles have a vertex in common. This has in general a negative answer, as the Petersen graph shows. However, there are some graph classes for which this question has a positive answer. As a common vertex of all longest cycles does not always exist, it is interesting to look for a set of vertices such that every longest cycle has at least one vertex in that set. Such a set is called a *longest cycle transversal*, or just a *transversal*. The minimum size of a transversal is denoted by $\mathrm{lct}(G)$. When we cannot determine $\mathrm{lct}(G)$ exactly, it is interesting to search for a good upper bound for it.

In what follows we consider a 2-connected graph G with n vertices. Thomassen [2] showed that $\mathrm{lct}(G) \leq \lceil n/3 \rceil$. This bound was improved by Rautenbach and Sereni [3], who proved that $\mathrm{lct}(G) \leq \lceil \frac{n}{3} - \frac{n^{2/3}}{36} \rceil$. Jobson *et al.* [4] showed that all longest cycles have a common intersection in dually chordal graphs, which includes doubly chordal, strongly chordal, and interval graphs. They also mention that their proof can be applied to show that all longest cycles intersect in split graphs. van Aardt *et al.* [5] showed that $\mathrm{lct}(G) \leq n/L$ for graphs whose longest cycles have length $L \leq 8$. Fernandes and the author [6] showed that $\mathrm{lct}(G) = 1$ if G is a 3-tree, and that $\mathrm{lct}(G) \leq 2$ if G is a partial 3-tree. To our knowledge, no results for general chordal graphs and graphs of bounded tree-width have been published. In this paper, we give results for

J. Gutiérrez—Research supported by FAPESP (Proc. 2015/08538-5).

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 558–571, 2018.
https://doi.org/10.1007/978-3-319-77404-6_41

lct(G) when G is chordal and when G has bounded tree-width. Moreover, we give a self-contained proof of the result mentioned by Jobson *et al.* [4] that all longest cycles intersect in 2-connected split graphs.

Similar results about transversals of longest paths have also been studied in the literature [7–14]. Particularly, Cerioli *et al.* [15,16] showed similar, but weaker, results for chordal and bounded tree-width graphs for paths instead of cycles. Other questions about intersection of longest cycles have also been raised by several authors [17–20].

In this paper, we prove the following:

– $\text{lct}(G) \leq \max\{1, \omega(G)-3\}$ for every 2-connected chordal graph G, where $\omega(G)$ is the size of a maximum clique in G (Sect. 3).
– $\text{lct}(G) \leq \text{tw}(G) - 1$ for every 2-connected graph G, where $\text{tw}(G)$ is the tree-width of G (Sect. 4).
– $\text{lct}(G) = 1$ for every 2-connected partial 3-tree G (Sect. 5).
– $\text{lct}(G) = 1$ for every 2-connected split graph G (Sect. 5).

In the next sections, we present these results in detail.

2 Preliminaries

In this section we introduce notations and previous results to be used in the rest of the paper. We also give an intuitive idea of our main technique. Some of the definitions given here are very similar to the ones used by Cerioli *et al.* [8,16].

2.1 Basic Concepts

All graphs considered in this paper are simple and 2-connected. (The later property is necessary to the fact that, when the graph is not 2-connected, two longest cycles may not intersect each other, and a minimum transversal can be arbitrary large.) Let C be a cycle in a graph G. We denote by $|C|$ the length of C, that is, the number of edges in C. Given a path C', we sometimes write C' to denote the vertices of C'. Given two paths C' and D' that share only at least one of their endpoints, we denote by $C' \cdot D'$ the union of C' and D'. For a pair of vertices $\{a, b\}$ in a cycle C, let C' and C'' be the paths such that $C = C' \cdot C''$ with $C' \cap C'' = \{a, b\}$. We refer to these two paths as the *ab-parts* of C. Given a cycle C that contains vertices a, b and c, and when the context is clear, we denote by C_{ab} the ab-part of C that does not contain c, by C_{bc} the bc-part of C that does not contain a, and by C_{ac} the ac-part of C that does not contain b.

Let S be a set of vertices in a graph G. Let C be a cycle or a path in G that does not contain all vertices of S and with a vertex not in S. We say that S *fences* C if all the vertices of $C - S$ are in a single component of $G - S$, otherwise we say that C *crosses* S. We say that C *k-intersects* S if $|C \cap S| = k$. Moreover, we also say that C k-intersects S at $C \cap S$. If C k-intersects S and is fenced by S, then we say that C is *k-fenced* by S. If C k-intersects S and crosses S, then we say that C *k-crosses* S. Similarly, we also can say that C *k-crosses* S

at $C \cap S$. If two cycles C and D are such that $C \cap S = D \cap S$, then we say they are *S-equivalent*, otherwise they are *S-nonequivalent*.

For a cycle or a path C, we denote by $\mathrm{Comp}_S(C)$ the set of vertices of the components of $G - S$ where $C - S$ lies. Two fenced cycles or paths C and D are *S-component-disjoint* if $\mathrm{Comp}_S(C) \cap \mathrm{Comp}_S(D) = \emptyset$. If S is clear from the context, we just say they are *component-disjoint*. Given a cycle C that 3-crosses S at $\{a, b, c\}$, we say that *a breaks C* if C_{ab} and C_{ac} are component-disjoint. Also, we say that a is a *C-breaking vertex*. If the context is clear, we just say it is a *breaking vertex*.

Let V_t be a set of vertices in G, and let C be a longest cycle in G. We say that V_t is *pulled by C* if all longest cycles that intersect V_t at $V_t \cap C$ (including C) are fenced by V_t. Given an integer $k \geq 0$, we say that V_t is *k-pulled* if V_t is pulled by a longest cycle C that k-intersects V_t. (Observe that $k = 0$ means that there exists at least one longest cycle that does not intersect V_t.) Finally, we say that V_t is *at most k-pulled* if it is k'-pulled for some $k' \in \{0, 1, \ldots, k\}$. Sometimes, we also say that V_t is *k-pulled by C* or *at most k-pulled* by C, where C is a corresponding cycle.

We use $L = L(G)$ for the length of a longest cycle in G. Also, we denote by $\omega(G)$ the size of a maximum clique in G.

2.2 Tree-Decomposition and Chordal Graphs

A *tree-decomposition* [21, p. 337] of a graph G is a pair $(T, \mathcal{V})$, conformed by a tree T and a collection $\mathcal{V} = \{V_t : t \in V(T)\}$ of *bags* $V_t \subseteq V(G)$, that satisfies the following three conditions:

(T1) $\bigcup_{t \in V(T)} V_t = V(G)$;
(T2) for every $uv \in E(G)$, there exists a bag V_t such that $u, v \in V_t$;
(T3) if a vertex v is in two different bags V_{t_1}, V_{t_2}, then v is also in any bag V_t such that t is on the (unique) path from t_1 to t_2 in T.

The *width* of $(T, \mathcal{V})$ is the number

$$\max\{|V_t| - 1 : t \in V(T)\},$$

and the *tree-width* $tw(G)$ of G is the minimum width of any tree-decomposition of G.

A graph is called *chordal* if every induced cycle has length three. Next we present some basic properties of tree decompositions for general and chordal graphs. We fix a 2-connected graph G and a tree-decomposition $(T, \mathcal{V})$ of G. Proposition 1 is due to Bodlaender [22]. The tree-decomposition mentioned in Proposition 2 is also called *clique tree* and it was introduced by Gavril [23].

Proposition 1. *If k is the tree-width of a graph G, then G has a tree-decomposition $(T, \mathcal{V})$ of width k such that $|V_t| = k + 1$ for every $t \in T$, and $|V_t \cap V_{t'}| = k$ for every $tt' \in T$.*

Proposition 2. *Every chordal graph has a tree-decomposition* $(T, \mathcal{V})$ *such that the bags of* $\mathcal{V}$ *are the maximal cliques of* G.

Given two different nodes t, t' of T, we denote by $\mathrm{Br}_t(t')$ the component of $T - t$ where t' lies. We say that such component is a *branch* of T at t and that the components of $T - t$ are the *branches* of T at t [24]. Similarly, for a vertex $v \notin V_t$, we denote by $\mathrm{Br}_t(v)$ the branch $\mathrm{Br}_t(t')$ of T at t such that $v \in V_{t'}$. Moreover, we can extend the notation and say that, if C is a path or a cycle fenced by V_t for some $t \in T$, then $\mathrm{Br}_t(C) = \mathrm{Br}_t(v)$, where v is a vertex of $C - V_t$. Propositions 3 to 5 are used to justify that the previous two definitions are coherent. The first two of them appear in the work of Heinz [24].

Proposition 3. *Let* t *be a node of* T *and* v *be a vertex of* G *such that* $v \notin V_t$. *Let* t' *and* t'' *be nodes of* T. *If* $v \in V_{t'} \cap V_{t''}$, *then* t' *and* t'' *are in the same branch of* T *at* t.

Proposition 4. *Let* u *and* v *be two vertices of* G, *and let* t *be a node of* T. *If* $u, v \notin V_t$, *and* u *and* v *are not separated by* V_t, *then* $\mathrm{Br}_t(u) = \mathrm{Br}_t(v)$.

Proposition 5. *Let* t *be a node of* T *and* C *be a path or cycle fenced by* V_t. *For every two vertices* u *and* v *in* $C - V_t$, $\mathrm{Br}_t(u) = \mathrm{Br}_t(v)$.

2.3 Proof Techniques

Our main technique resembles the Helly Property on trees. It is known that, in a tree, a collection of pairwise intersecting subtrees have a vertex in common. (As a corollary of that property, all longest paths intersect in trees.) There are several proofs of that fact, we focus on the following one. If we suppose by contradiction that it is not the case that there exists such a vertex in common, then it means that, for every vertex, there exists a subtree that does not contain that particular vertex. That subtree fits in exactly one component of the forest that remains when we remove such a vertex. Thus, we can add an arc from that vertex to the corresponding neighbor of it. In this way, by analyzing a maximal directed path, we find a contradiction by observing the last arc of this path.

So, our main task is to resemble this idea with the help of a tree-decomposition of the graph. It is not the first time this idea is used. According to Diestel [21], a *bramble* is a set of sets of vertices in a graph such that every pair of such sets either intersect each other or the graph contains an edge between them. (Observe that the set of longest cycles of a 2-connected graph is a bramble.) Theorem 12.3.9 of the same book [21] directly implies that, if a graph has tree-width tw, then there exists a longest cycle transversal of size $\mathrm{tw} + 1$. (This was observed by Rautenbach and Sereni [3] in Proposition 2.6 but for paths instead of cycles.) As chordal graphs have tree-width $\omega - 1$, we can easily conclude that chordal graphs have a longest cycle transversal of size ω. (This was also observed before, by Balister *et al.* [7], for paths instead of cycles.)

Our main target in this paper is to improve these bounds. For that, as we said, we try to resemble the idea for trees as follows. First, we focus on an arbitrary bag of the tree-decomposition (if the graph is chordal, we are focusing on a maximal clique). Then, we try to direct the corresponding node of this bag towards a neighbor of the node in the tree-decomposition. To do that, we need a longest cycle that lies into the corresponding branch (in fact, we will need a little more, that is why we introduced the definition of pulled). The main difficulty is to prove that such a cycle exists. And, as we will see in our next lemmas, it is very unlikely that a set of longest cycles that crosses a bag at the same time, and intersect the bag in a little quantity of vertices, can coexist. Hence, we will obtain at least one fenced cycle in order to do the orientation. In the next sections we will see the details of how this cycle is obtained. After that, it remains to analyze the last arc of this orientation to obtain a final contradiction. That will happen in the proof of our main theorems.

3 Chordal Graphs

The purpose of this section is to prove an upper bound for lct in any 2-connected chordal graph. We start by proving a property and a lemma valid for all 2-connected graphs.

Proposition 6. *Let G be a 2-connected graph with a clique V_t. Then every two longest cycles that 2-cross V_t are V_t-equivalent.*

Proof. Suppose by contradiction that there exists two longest cycles C and D that 2-cross V_t and $C \cap V_t \neq D \cap V_t$. Let $C \cap V_t = \{a, b\}$ and $D \cap V_t = \{c, d\}$. We may assume that either $\{a, b\}$ and $\{c, d\}$ are disjoint or $a \neq b = d \neq c$. Let C' and C'' be the two ab-parts of C. Let D' and D'' be the two cd-parts of D. As both C and D cross V_t, we have that any of $\{C', C''\}$ is component-disjoint from at least one of $\{D', D''\}$, and that any of $\{D', D''\}$ is component-disjoint from at least one of $\{C', C''\}$. Hence, we may assume, without loss of generality, that C' is component-disjoint from D', and that C'' is component-disjoint from D''. Hence, $C' \cdot ac \cdot D' \cdot db$ and $C'' \cdot ac \cdot D'' \cdot db$ are cycles, one of them longer than L, a contradiction. □

Lemma 1. *Let G be a 2-connected graph with a clique V_t such that $|V_t| \geq 5$. If there exists a longest cycle that 2-crosses V_t, then either* $\mathrm{lct}(G) \leq |V_t| - 3$ *or V_t is at most 3-pulled.*

Proof. Let C be a longest cycle that 2-crosses V_t. Suppose that $C \cap V_t = \{a, b\}$. Let S be a subset of vertices of V_t of size $|V_t| - 3$ that contains $\{a, b\}$. Such a subset exists, because we are assuming that $|V_t| \geq 5$. Suppose that $\mathrm{lct}(G) > |V_t| - 3$. Hence, there exists a longest cycle $\bar{D}$ that does not contain any vertex of S. This implies that $\bar{D}$ intersects V_t at most three times, and that $\bar{D}$ does not contain any of $\{a, b\}$. If $\bar{D}$ intersects V_t at most once, it is fenced by V_t, hence V_t is 0-pulled or 1-pulled and we are done. Thus, we may assume that $\bar{D}$ intersects V_t at least twice. Suppose that $\bar{D}$ 2-intersects V_t. Recall that C 2-crosses V_t at $\{a, b\}$. Hence, by Proposition 6, all longest cycles that 2-cross V_t, intersect V_t at $\{a, b\}$.

Thus, all longest cycles that intersect V_t at $V_t \cap \bar{D}$, particularly $\bar{D}$, are fenced by V_t. Hence, V_t is 2-pulled by $\bar{D}$ and we are done.

Now suppose that $\bar{D}$ 3-intersects V_t. Let $\bar{D} \cap V_t = \{c, d, f\}$. Suppose for a moment that there exists a longest cycle D that crosses V_t at $\{c, d, f\}$. As C and D cross V_t, any of $\{D_{cd}, D_{df}, D_{cf}\}$ is component-disjoint with at least one of $\{C', C''\}$, and any of $\{C', C''\}$ is component-disjoint with at least one $\{D_{cd}, D_{df}, D_{cf}\}$. Thus, we may assume, without loss of generality, that C' is component-disjoint from D_{cd} and D_{cf}, and that C'' is component-disjoint from D_{df}. But then, $C' \cdot ad \cdot D_{dc} \cdot D_{cf} \cdot fb$ and $C'' \cdot ad \cdot D_{df} \cdot fb$ are both cycles, one of them longer than L, a contradiction. Hence, every cycle that intersect V_t at $\{c, d, f\} = \bar{D} \cap V_t$ is fenced by V_t. We conclude that V_t is 3-pulled by $\bar{D}$. $\square$

We can extend the previous lemma, but just for chordal graphs. Before that, we state some useful properties. Proposition 7 appears in the book of Diestel [21] as Lemma 12.3.1.

Proposition 7. *Let $tt' \in E(T)$. Then $V_t \cap V_{t'}$ separates, in G, a vertex in* $\mathrm{Br}_t(t')$ *from a vertex in* $\mathrm{Br}_{t'}(t)$.

Proposition 8. *Let $t \in V(T)$. Let C' be a path 2-fenced by V_t such that* $\mathrm{Br}_t(C') = \mathrm{Br}_t(t')$, *where $tt' \in E(T)$. Then $V_t \cap C' \subseteq V_{t'}$.*

Proposition 9. *Let $t \in V(T)$. Let C and D be two paths or cycles fenced by V_t. If* $\mathrm{Comp}_{V_t}(C) = \mathrm{Comp}_{V_t}(D)$, *then* $\mathrm{Br}_t(C) = \mathrm{Br}_t(D)$.

Proposition 10. *Let $t \in V(T)$. Let $\mathscr{C}$ be a collection of paths 2-fenced by V_t. If $\bigcup_{C' \in \mathscr{C}} C' \cap V_t = V_t$, then there exists two paths C' and D' in $\mathscr{C}$, such that* $\mathrm{Br}_t(C') \neq \mathrm{Br}_t(D')$ *and* $\mathrm{Comp}_{V_t}(C') \neq \mathrm{Comp}_{V_t}(D')$,

Proof. Suppose that $\bigcup_{C' \in \mathscr{C}} C' \cap V_t = V_t$. By Proposition 9, it suffices to show that there exists two paths C' and D' in $\mathscr{C}$ such that $\mathrm{Br}_t(C') \neq \mathrm{Br}_t(D')$. Suppose by contradiction that for every pair of paths C' and D' in $\mathscr{C}$, we have that $\mathrm{Br}_t(C') = \mathrm{Br}_t(D')$. Let $tt' \in E(T)$ such that $\mathrm{Br}_t(C') = \mathrm{Br}_t(t')$. By Proposition 8, $V_t \cap C' \subseteq V_{t'}$ for every $C' \in \mathscr{C}$. Hence, $V_t \subseteq V_{t'}$. As $V_t \neq V_{t'}$, we have a contradiction to the maximality of V_t. $\square$

Lemma 2. *Let G be a 2-connected chordal graph with a maximal clique V_t such that $|V_t| \leq 4$. If there exists a longest cycle that 2-crosses V_t, then either* $\mathrm{lct}(G)= 1$ *or V_t is at most 3-pulled.*

Proof. Let $(T, \mathcal{V})$ be a tree-decomposition of G as in Proposition 2. We consider the following two cases.

Case 1: $|V_t| = 4$**.** Let $V_t = \{a, b, c, d\}$. Let C be a longest cycle that 2-crosses V_t at ab. Let $\{C', C''\}$ be the two ab-parts of C. Suppose that $\mathrm{lct}(G) > 1$. Then, there exists a longest cycle $\bar{D}$ that does not contain a. If $\bar{D}$ intersects V_t at most once, then we are done. Hence, we may assume that $\bar{D}$ intersects V_t at least twice. Suppose that $\bar{D}$ 2-intersects V_t. Recall that C 2-crosses V_t at $\{a, b\}$. Hence, by Proposition 6, all longest cycles that 2-cross V_t, intersect V_t at $\{a, b\}$.

Thus, all longest cycles that intersect V_t at $V_t \cap \bar{D}$, particularly $\bar{D}$, are fenced by V_t. Hence, V_t is 2-pulled by $\bar{D}$ and we are done.

Thus, $\bar{D}$ 3-intersects V_t. Let $\bar{D} \cap V_t = \{b, c, d\}$. Suppose for a moment that there exists a longest cycle, called it D, that crosses V_t at $\{b, c, d\}$. Suppose also for a moment that b breaks D. That is, D_{bc} and D_{bd} are component-disjoint. Then, as C' is also component-disjoint from C'', we may assume, without loss of generality, that C' is component-disjoint from D_{bc} and that C'' is component-disjoint from D_{bd}. Also, D_{cd} is component-disjoint with at least one of $\{C', C''\}$, suppose it is C'. Then $C' \cdot D_{bc} \cdot D_{cd} \cdot da$ and $C'' \cdot D_{bd} \cdot da$ are cycles, a contradiction. Thus, b does not break D. That is, $\mathrm{Comp}_{V_t}(D_{bc}) = \mathrm{Comp}_{V_t}(D_{bd})$, and, by Proposition 9,

$$\mathrm{Br}_t(D_{bc}) = \mathrm{Br}_t(D_{bd}). \tag{1}$$

Observe also that, by Proposition 10,

$$\mathrm{Br}_t(D_{cd}) \neq \mathrm{Br}_t(C'), \tag{2}$$

and

$$\mathrm{Br}_t(D_{cd}) \neq \mathrm{Br}_t(C''). \tag{3}$$

Suppose for a moment that $\mathrm{Br}_t(D_{bc}) = \mathrm{Br}_t(C')$. Then, by (1), $\mathrm{Br}_t(D_{bc}) = \mathrm{Br}_t(D_{bd}) = \mathrm{Br}_t(C')$. Let $tt' \in E(T)$ such that $\mathrm{Br}_t(C') = \mathrm{Br}_t(t')$. By Proposition 8, we have that $V_t \cap C' = \{a, b\} \subseteq V_{t'}$, that $V_t \cap D_{cd} = \{c, d\} \subseteq V_{t'}$, and that $V_t \cap D_{bd} = \{b, d\} \subseteq V_{t'}$. So, $V_t = \{a, b, c, d\} \subseteq V_{t'}$, a contradiction to the maximality of V_t. Hence, by (1),

$$\mathrm{Br}_t(D_{bc}) = \mathrm{Br}_t(D_{bd}) \neq \mathrm{Br}_t(C'). \tag{4}$$

And, by a similar argument,

$$\mathrm{Br}_t(D_{bc}) = \mathrm{Br}_t(D_{bd}) \neq \mathrm{Br}_t(C''). \tag{5}$$

But then, by (2), (3), (4) and (5), C and D only intersect at b, a contradiction to the fact that G is 2-connected. Hence, every cycle that intersects V_t at $\{b, c, d\} = \bar{D} \cap V_t$ is fenced by V_t. We conclude that V_t is 3-pulled by $\bar{D}$.

Case 2: $|V_t| \leq 3$. Let $V_t = \{a, b, c\}$. Let C be a longest cycle that 2-crosses V_t at ab. Suppose that $\mathrm{lct}(G) > 1$. Then, there exists a longest cycle $\bar{D}$ that does not contain a. If $\bar{D}$ intersects V_t at most once, then we are done. Hence, we may assume that $\bar{D}$ 2-intersects V_t. Recall that C 2-crosses V_t at $\{a, b\}$. Hence, by Proposition 6, all longest cycles that 2-cross V_t, intersect V_t at $\{a, b\}$. Thus, all longest cycles that intersect V_t at $V_t \cap \bar{D}$, particularly $\bar{D}$, are fenced by V_t. Hence, V_t is 2-pulled by $\bar{D}$ and we are done. □

Now we study how longest cycles that 3-cross V_t behave. Remember that, given a longest cycle C that 3-crosses V_t at $\{a, b, c\}$, we say a is a C-breaking vertex if C_{ab} and C_{ac} are V_t-component-disjoint.

Proposition 11. *Let G be a graph. Let C be a cycle in G. Let V_t be a set of vertices of G. Let Δ be a triangle in V_t. If C 3-crosses V_t at Δ, then Δ has at least two C-breaking vertices.*

Lemma 3. *Let G be a 2-connected graph with a clique V_t such that $|V_t| \geq 5$. If for every triangle Δ in V_t there exists a longest cycle that 3-intersects V_t at Δ, then V_t is pulled by at least one of such cycles.*

Proof. Suppose by contradiction that V_t is not pulled by any of such cycles. Then, for every triangle Δ in V_t there exists a longest cycle that 3-crosses V_t at Δ. By Proposition 11, for every Δ in V_t, Δ have at least two breaking vertices. As there are $\binom{|V_t|}{3}$ triangles in V_t, by pigeonhole principle, there exists a vertex $v \in V_t$ such that v is a breaking vertex for at least $\frac{(|V_t|-1)(|V_t|-2)}{3}$ of the triangles incident to v. As $|V_t| \geq 5$, there exists two edge-disjoint triangles incident to v such that v is a breaking vertex for both of them. Let vab and vcd be such triangles, and let C and D be the corresponding cycles respectively. As v breaks both C and D, without loss of generality we may assume that C_{va} and D_{vc} are component-disjoint and that C_{vb} and D_{vd} are component-disjoint. Also, C_{ab} is component-disjoint with at least one of $\{D_{vc}, D_{vd}\}$, and D_{cd} is component-disjoint with at least one of $\{C_{va}, C_{vb}\}$. Without loss of generality, we may suppose that C_{ab} is component-disjoint from D_{vd}. If D_{cd} is component-disjoint from C_{va} then $D_{vc} \cdot D_{cd} \cdot da \cdot C_{av}$ and $D_{dv} \cdot C_{vb} \cdot C_{ba} \cdot ad$ are cycles, a contradiction. So $\text{Comp}_{V_t}(D_{cd}) = \text{Comp}_{V_t}(C_{va})$ and D_{cd} is component-disjoint from C_{vb}. If C_{ab} is component-disjoint from D_{cd} then $C_{va} \cdot ac \cdot D_{ca}$ and $C_{vb} \cdot C_{ba} \cdot ac \cdot D_{cd} \cdot D_{dv}$ are cycles, a contradiction. So, $\text{Comp}_{V_t}(C_{ab}) = \text{Comp}_{V_t}(D_{cd})$. As $\text{Comp}_{V_t}(D_{cd}) = \text{Comp}_{V_t}(C_{va})$, we conclude that $\text{Comp}_{V_t}(C_{ab}) = \text{Comp}_{V_t}(C_{va})$. And as C_{va} and D_{vc} are component-disjoint, we conclude that C_{ab} and D_{vc} are component-disjoint. Then, $C_{va} \cdot C_{ab} \cdot bc \cdot D_{cv}$ and $C_{vb} \cdot bc \cdot D_{cd} \cdot D_{dv}$ are both cycles, one of them longer than L, again a contradiction. □

We can extend the previous lemma, but just for chordal graphs.

Lemma 4. *Let G be a 2-connected chordal graph with a maximal clique V_t such that $|V_t| = 4$. If for every triangle Δ in V_t there exists a longest cycle that 3-intersects V_t at Δ, then either* $\text{lct}(G) = 1$ *or V_t is at most 3-pulled.*

Proof. Let $(T, \mathcal{V})$ be a tree-decomposition of G as in Proposition 2. Suppose that V_t is not at most 3-pulled. Then, for every triangle Δ in V_t there exists a longest cycle that 3-crosses V_t at Δ. By Proposition 11, for every Δ in V_t, Δ has at least two breaking vertices. As $|V_t| = 4$, there are four such triangles. By pigeonhole principle, there exists a vertex $a \in V_t$ such that a is a breaking vertex for at least two of the triangles incident to a. Let abd and acd be these two triangles. Let C and D be the corresponding longest cycles respectively. Hence,

$$\text{Comp}_{V_t}(C_{ab}) \neq \text{Comp}_{V_t}(C_{ad}) \tag{6}$$

and

$$\text{Comp}_{V_t}(D_{ac}) \neq \text{Comp}_{V_t}(D_{ad}). \tag{7}$$

Also, by Proposition 10,

$$\mathrm{Comp}_{V_t}(C_{ab}) \neq \mathrm{Comp}_{V_t}(D_{cd}) \tag{8}$$

and

$$\mathrm{Comp}_{V_t}(C_{bd}) \neq \mathrm{Comp}_{V_t}(D_{ac}). \tag{9}$$

By (6) and (7), either $\mathrm{Comp}_{V_t}(C_{ab}) \neq \mathrm{Comp}_{V_t}(D_{ad})$ and $\mathrm{Comp}_{V_t}(C_{ad}) \neq \mathrm{Comp}_{V_t}(D_{ac})$, or $\mathrm{Comp}_{V_t}(C_{ab}) \neq \mathrm{Comp}_{V_t}(D_{ac})$ and $\mathrm{Comp}_{V_t}(C_{ad}) \neq \mathrm{Comp}_{V_t}(D_{ad})$. If the first case is true, then $C_{ba} \cdot D_{ad} \cdot D_{dc} \cdot cb$ and $D_{ac} \cdot cb \cdot C_{bd} \cdot C_{da}$ are cycles, one of them longer than L, a contradiction. If the second case is true, then $\mathrm{Comp}_{V_t}(C_{bd}) \neq \mathrm{Comp}_{V_t}(D_{cd})$. Indeed, suppose for a moment that $\mathrm{Comp}_{V_t}(C_{bd}) = \mathrm{Comp}_{V_t}(D_{cd})$. Then, by Proposition 10, $\mathrm{Comp}_{V_t}(C_{ad}) \neq \mathrm{Comp}_{V_t}(D_{cd})$ and $\mathrm{Comp}_{V_t}(C_{bd}) \neq \mathrm{Comp}_{V_t}(D_{ad})$, so $C_{ba} \cdot C_{ad} \cdot D_{dc} \cdot cb$ and $D_{ca} \cdot C_{ad} \cdot D_{db} \cdot bc$ are cycles, one of them longer than L, a contradiction. Hence, $\mathrm{Comp}_{V_t}(C_{bd}) \neq \mathrm{Comp}_{V_t}(D_{cd})$. Thus, $C_{ad} \cdot D_{ad}$ and $C_{ab} \cdot C_{bd} \cdot D_{dc} \cdot D_{ca}$ are cycles. But then, $C_{ad} \cdot D_{ad}$ is a longest cycle that 2-crosses V_t, and we conclude the proof by applying Lemma 2. □

The following lemma synthesizes the previous four lemmas.

Lemma 5. *Let G be a 2-connected chordal graph with a maximal clique V_t. Either $\mathrm{lct}(G) \leq \max\{1, |V_t| - 3\}$ or V_t is at most 3-pulled.*

Proof. Suppose that $\mathrm{lct}(G) > \max\{1, |V_t| - 3\}$. Then, for every set of $\max\{1, |V_t| - 3\}$ vertices in V_t, there exists a longest cycle that does not contain any vertex of that set. If one of these cycles intersects V_t at most once, then we are done. Hence, we may assume that every such cycle intersects V_t at least twice. Suppose for a moment that one of these cycles, called it C, 2-intersects V_t. If C, or any longest cycle equivalent to C, 2-crosses V_t, then we finish by Lemmas 1 and 2. So, all longest cycles equivalent to C are fenced by V_t and V_t is 2-pulled by C. Thus, as $|V_t| - \max\{1, |V_t| - 3\} \leq 3$, every such cycle 3-intersects V_t and we finish by Lemmas 3 and 4. □

Proposition 12. *If C is a cycle fenced by V_t for some $t \in T$, then there exists a neighbor t' of t in T such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$.*

Proposition 13. *If every clique V_t in G is at most k-pulled, then there exist two longest cycles C and D fenced by V_t, and an edge tt' of T such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$, $\mathrm{Br}_{t'}(D) = \mathrm{Br}_{t'}(t)$, V_t is at most k-pulled by C, and $V_{t'}$ is at most k-pulled by D.*

Proof. We define a digraph T' as follows. (i) $V(T) = V(T')$ and (ii) $tt' \in E(T')$ if and only if $tt' \in E(T)$ and there exists a longest cycle C in G fenced by V_t such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$ and V_t is at most k-pulled by C. Let t be an arbitrary node of T. As V_t is at most k-pulled, there exists a longest cycle C fenced by V_t such that V_t is at most k-pulled by C. By Proposition 12, there exists a neighbor t' of t in T such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$. Hence, every node in T' has outdegree at least one. Let tt' be the last arc of a maximal directed path in T'. As T is a tree, $t't$ is also an arc in T', which implies that there exist two longest cycles C and D in G such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$, $\mathrm{Br}_{t'}(D) = \mathrm{Br}_{t'}(t)$, V_t is at most k-pulled by C, and $V_{t'}$ is at most k-pulled by D.

Proposition 14. *Let tt' be an edge of T and let u and v be two vertices of G with $v \notin V_t$. If $\mathrm{Br}_t(v) = \mathrm{Br}_t(t')$ and $u \in V_t \setminus V_{t'}$, then u and v are not adjacent.*

Finally, we obtain our main result.

Theorem 1. *For every 2-connected chordal graph G, $\mathrm{lct}(G) \leq \max\{1, \omega(G)-3\}$.*

Proof. Suppose by contradiction that $\mathrm{lct}(G) > \max\{1, \omega(G) - 3\}$. Let $(T, \mathcal{V})$ be a tree-decomposition of G as in Proposition 2. By Lemma 5, every clique V_t in G is at most 3-pulled. Thus, by Proposition 13, there exist two longest cycles C and D in G such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$, $\mathrm{Br}_{t'}(D) = \mathrm{Br}_{t'}(t)$, V_t is at most 3-pulled by C, and $V_{t'}$ is at most 3-pulled by D.

Note that the bags containing vertices of C are only in $\mathrm{Br}_t(t') \cup \{t\}$, and the bags containing vertices of D are only in $\mathrm{Br}_{t'}(t) \cup \{t'\}$. As $\mathrm{Br}_t(t')$ and $\mathrm{Br}_{t'}(t)$ are disjoint, $C \cap D \subseteq V_t \cup V_{t'}$. Let u be a vertex such that $u \in V_t \setminus V_{t'}$. Suppose for a moment that C contains u and let v be a neighbor of u in C. By Proposition 14, $\mathrm{Br}_t(v) \neq \mathrm{Br}_t(t')$ so $v \in V_t$. This implies that uv is an edge in V_t and, as V_t is a clique, C contains all vertices of V_t, contradicting the fact that C is fenced. So C does not contain vertices in $V_t \setminus V_{t'}$. By a similar argument, D does not contain vertices in $V_{t'} \setminus V_t$. Thus $C \cap D \subseteq V_t \cap V_{t'}$. As G is 2-connected, both C and D intersects V_t and $V_{t'}$ at least twice, respectively.

Suppose for a moment that $|V_t \cap V_{t'}| \leq \omega(G) - 2$. Then, as $\mathrm{lct}(G) > \max\{1, \omega(G) - 3\}$, there exists a longest cycle R that contains at most one vertex of $V_t \cap V_{t'}$. But then R intersect with one of $\{C, D\}$ at most once, a contradiction to the fact that G is 2-connected. Hence, $|V_t \cap V_{t'}| \geq \omega(G) - 1$. Thus, as both V_t and $V_{t'}$ are maximal (and different), we conclude that $|V_t| = |V_{t'}| = \omega(G)$ and that $|V_t \cap V_{t'}| = \omega(G) - 1$. Moreover, if $\omega(G) \leq 3$, then $\mathrm{lct}(G) > 1$ and there exists a longest cycle R that contains at most one vertex of $V_t \cap V_{t'}$, again a contradiction. We conclude that $\omega(G) \geq 4$ and that $\mathrm{lct}(G) > \omega(G) - 3$. Remember that C and D intersect, respectively V_t and $V_{t'}$, at least twice and at most three times. Let $\{u\} = V_t \setminus V_{t'}$ and $\{w\} = V_{t'} \setminus V_t$. We divide the rest of the proof in three cases.

Case 1: Both C and D 3-intersect V_t and $V_{t'}$, respectively. Let $C \cap V_t = \{x, y, z\}$. Consider the case when $D \cap V_t = \{x, y, z\}$. By pigeonhole principle, we may assume, without loss of generality, that $u \notin C_{xy}$ and that $w \notin D_{xy}$. As $(C - C_{xy}) \cdot D_{xy}$ and $(D - D_{xy}) \cdot C_{xy}$ are cycles, $|C_{xy}| = |D_{xy}|$ and both are longest cycles. Hence, $(C - C_{xy}) \cdot D_{xy}$ is a longest cycle that 3-crosses V_t at $C \cap V_t$, a contradiction to the fact that V_t is pulled by C. Now suppose that $D \cap V_t = \{y, z, w\}$, with $w \neq x$. Then, $C_{yz} \cdot C_{zx} \cdot xw \cdot D_{wy}$ and $D_{yz} \cdot D_{zw} \cdot wx \cdot C_{xy}$ are cycles, one of them longer than L, a contradiction.

Case 2: Both C and D 2-intersect V_t and $V_{t'}$, respectively. As G is 2-connected, we may assume that $C \cap V_t = D \cap V_t = \{x, y\}$. Let C' and C'' be the two xy-parts of C. Let D' and D'' be the two xy-parts of D. As $(C - C') \cdot D', (C - C') \cdot D'', (D - D') \cdot C'$ and $(D - D') \cdot C''$ are cycles, $|C'| = |C''| = |D'| = |D''| = L/2$. Without loss of generality we may assume that $u \notin D'$. Hence, $D' \cdot C'$ is a longest cycle that 2-crosses V_t at $C \cap V_t$, a contradiction to the fact that V_t is pulled by C.

Case 3: C **3-intersects** V_t **and** D **2-intersects** $V_{t'}$ As G is 2-connected, we may assume that $C \cap V_t = \{x, y, z\}$ and that $D \cap V_t = \{x, y\}$. Let D' and D'' be the two xy-parts of D. Without loss of generality, we may assume that $u \notin D'$. Hence, $(C - C_{xy}) \cdot D'$ is a longest cycle that 3-crosses V_t at $C \cap V_t$, a contradiction to the fact that V_t is pulled by C. □

The previous theorem implies the following results.

Corollary 1. *All longest cycles intersect in 2-trees, 3-trees, and in 2-connected chordal planar graphs.*

4 Graphs of Bounded Tree-Width

In this section, we prove an upper bound for $\text{lct}(G)$ when G has bounded tree-width.

Lemma 6. *Let G be a 2-connected graph with tree-width at least two. Let $(T, \mathcal{V})$ be a tree-decomposition of G as in Proposition 1. Let $V_t \in \mathcal{V}$. If $\text{lct}(G) > |V_t| - 2$, then V_t is at most 2-pulled.*

Proof. Suppose that $\text{lct}(G) > |V_t| - 2$. This implies that no subset of V_t of size $|V_t| - 2$ is a longest cycle transversal in G. Thus, for every such subset, there exists a longest cycle that does not contain any vertex of it. If any of these cycles intersects V_t at most once, we are done. Hence, every such cycle 2-intersects V_t. That is, for every pair of vertices in V_t, there exists a longest cycle that 2-intersects V_t at such pair. If V_t is pulled by any such cycle, we are done. Hence, for every pair of vertices in V_t, there exists a longest cycle that 2-crosses V_t at such pair. Let $\{a, b\} \subset V_t$ and let C be a longest cycle that 2-crosses V_t at $\{a, b\}$. Let C' and C'' be the two ab-parts of C. By Proposition 12, there exists two nodes t', t'', neighbors of t in T, such that $\text{Br}_t(C') = \text{Br}_t(t')$ and $\text{Br}_t(C'') = \text{Br}_t(t'')$, where possibly $t' = t''$.

Case 1: $V_t \cap V_{t'} = V_t \cap V_{t''}$**.** As $|V_t \cap V_{t'}| = |V_t| - 1$, there exists a vertex $x \in V_t \setminus V_{t'} = V_t \setminus V_{t''}$. Let y be an arbitrary vertex in V_t different from x. Let D be a longest cycle that 2-crosses V_t at $\{x, y\}$. Then, as C and D intersect each other in at least two vertices, there exists a xy-part of D, called it D', such that either $\text{Br}_t(D') = \text{Br}_t(t')$ or $\text{Br}_t(D') = \text{Br}_t(t'')$. If the former is true, then, by Proposition 8, $\{x, y\} = V_t \cap D' \subseteq V_{t'}$, a contradiction to the fact that $x \in V_t \setminus V_{t'}$. If the later is true, then, by Proposition 8, $\{x, y\} = V_t \cap D' \subseteq V_{t''}$, a contradiction to the fact that $x \in V_t \setminus V_{t''}$.

Case 2: $V_t \cap V_{t'} \neq V_t \cap V_{t''}$**.** As $|V_t \cap V_{t'}| = |V_t \cap V_{t''}|$, there exists a vertex $x \in (V_t \cap V_{t'}) \setminus V_{t''}$ and a vertex $y \in (V_t \cap V_{t''}) \setminus V_{t'}$. Let D be a longest cycle that 2-crosses V_t at $\{x, y\}$. Then, as C and D intersect in at least two vertices, there exists a xy-part of D, called it D', such that $\mathrm{Br}_t(D') = \mathrm{Br}_t(t')$ or $\mathrm{Br}_t(D') = \mathrm{Br}_t(t'')$. If the former is true, then by Proposition 8, $\{x, y\} = V_t \cap D' \subseteq V_{t'}$, a contradiction to the fact that $y \in (V_t \cap V_{t''}) \setminus V_{t'}$. If the later is true, then by Proposition 8, $\{x, y\} = V_t \cap D' \subseteq V_{t''}$, a contradiction to the fact that $x \in (V_t \cap V_{t'}) \setminus V_{t''}$. □

Proposition 15. *Let tt' be an edge of T and let C be a cycle in G. If C has a vertex in $V_t \setminus V_{t'}$ and a vertex in $\mathrm{Br}_t(t')$, then $|C \cap V_t| \geq 3$.*

Theorem 2. *For every 2-connected graph G, $lct(G) \leq \mathrm{tw}(G) - 1$.*

Proof. As G is 2-connected, it is not a forest, so $\mathrm{tw}(G) \geq 2$. Let $(T, \mathcal{V})$ be a tree-decomposition of G as in Proposition 1. Suppose by contradiction that $\mathrm{lct}(G) > \mathrm{tw}(G) - 1$. Let $t \in V(T)$. Then, as $|V_t| = \mathrm{tw}(G) + 1$, by Lemma 6, V_t is at most 2-pulled. Thus, by Proposition 13, there exist two longest cycles C and D in G such that $\mathrm{Br}_t(C) = \mathrm{Br}_t(t')$, $\mathrm{Br}_{t'}(D) = \mathrm{Br}_{t'}(t)$, V_t is at most 2-pulled by C, and $V_{t'}$ is at most 2-pulled by D.

Note that the bags containing vertices of C are only in $\mathrm{Br}_t(t') \cup \{t\}$, and that the bags containing vertices of D are only in $\mathrm{Br}_{t'}(t) \cup \{t'\}$. As $\mathrm{Br}_t(t')$ and $\mathrm{Br}_{t'}(t)$ are disjoint, $C \cap D \subseteq V_t \cup V_{t'}$. Let u be the vertex such that $\{u\} = V_t \setminus V_{t'}$. As C is fenced, by definition, there exists a vertex v in C such that v is not in V_t. Suppose for a moment that C contains u. By Proposition 15, $|C \cap V_t| \geq 3$. But, as V_t is at most 2-pulled by C, C intersects V_t at most twice, a contradiction.

So C does not contain the only vertex in $V_t \setminus V_{t'}$. By a similar argument, D does not contain the only vertex in $V_{t'} \setminus V_t$. Thus $C \cap D \subseteq V_t \cap V_{t'}$. As G is 2-connected,

$$2 \leq |C \cap D| \ = |C \cap D \cap V_t \cap V_{t'}|.$$

This implies that $|C \cap V_t| \geq 2$ and $|D \cap V_{t'}| \geq 2$, therefore, as C and D are given by Lemma 6, C 2-intersects V_t and D 2-intersects $V_{t'}$. As G is 2-connected, we may assume that $C \cap V_t = D \cap V_t = \{x, y\}$. Let C' and C'' be the two xy-parts of C. Let D' and D'' be the two xy-parts of D. As $|C \cap D| = 2$, we can conclude that $|C'| = |C''| = |D'| = |D''|$. And, as $|V_t \cap V_{t'}| = |V_{t'}| - 1$, at least one of $\{D', D''\}$, suppose it is D', does not contain the only vertex of $V_{t'} \setminus V_t$. Hence, $C' \cdot D'$ is a longest cycle that 2-crosses V_t at $\{x, y\} = C \cap V_t$, a contradiction to the fact that V_t is pulled by C. □

The previous theorem implies the following results.

Corollary 2. *All longest cycles intersect in 2-connected series-parallel graphs.*

Planar graphs do not have bounded tree-width. However, Fomin and Thilikos [25] showed that a planar graph G on n vertices has tree-width at most $3.182\sqrt{n}$. More generally, Alon, Seymour, and Thomas [26] showed that any K_r-minor free graph on n vertices has tree-width at most $r^{1.5}\sqrt{n}$. Hence, we have the following corollaries.

Corollary 3. *For every 2-connected planar graph G on n vertices,* $\mathrm{lct}(G) < 3.182\sqrt{n}$*.*

Corollary 4. *For every 2-connected K_r-minor free graph G,* $\mathrm{lct}(G) < r^{1.5}\sqrt{n}$*.*

5 Partial 3-Trees and Split Graphs

By Theorem 2 (Sect. 4), every partial 3-tree has a longest cycle transversal of size at most 2. We show that in fact there is a transversal of size one, that is, all longest cycles intersect in 2-connected partial 3-trees. As the proof is longer than the others, we choose to omit it. The difficulty arises when trying to analyze longest cycles that 3-intersects a bag. But generally speaking, the main idea of the other proofs is maintained. We also give a a self-contained proof of the result given by Jobson *et al.* [4] that all longest cycles intersect in split graphs.

Theorem 3. *If G be a 2-connected partial 3-tree, then* $\mathrm{lct}(G) = 1$*.*

Theorem 4. *If G is a 2-connected split graph then $lct(G) = 1$.*

Proof sketch: We say that a graph G is *minimal-lct* if $lct(G) > 1$ but for every proper subgraph H of G, $lct(H) = 1$. It is easy to show that If $G = (V, E)$ is a minimal-lct split graph with a splitting (K, S), then, for every edge $uv \in E$ such that $u \in K$ and $v \in S$, there is a longest cycle in G that contains uv. By analyzing such a minimal counterexample we can obtain the contradiction we want. □

6 Final Remarks

In this paper we showed upper bounds for the minimum size of a set of vertices that intersects all longest cycles in a 2-connected graph. A natural question asks for lower instead of upper bounds. It is known that it is not always the case that all longest cycles intersect in general 2-connected graphs, so $\mathrm{lct}(G) > 1$ if G is arbitrary. The question of whether $\mathrm{lct}(G) = 1$ when G is chordal is still open. For bounded tree-width graphs, there exists a 2-connected graph G given by Thomassen on 15 vertices [27], with tree-width four and $\mathrm{lct}(G) = 2$. Hence, by Theorem 2, we conclude that $\mathrm{lct}(G) \in \{2, 3\}$ in partial 4-trees.

References

1. Grötschel, M.: On intersections of longest cycles. In: Bollobás, B. (ed.) Graph Theory and Combinatorics, pp. 171–189 (1984)
2. Thomassen, C.: Hypohamiltonian graphs and digraphs. In: Alavi, Y., Lick, D.R. (eds.) Theory and Applications of Graphs. LNM, vol. 642, pp. 557–571. Springer, Heidelberg (1978). https://doi.org/10.1007/BFb0070410
3. Rautenbach, D., Sereni, J.S.: Transversals of longest paths and cycles. SIAM J. Discret. Math. **28**(1), 335–341 (2014)

4. Jobson, A., Kzdy, A., Lehel, J., White, S.: Detour trees. Discret. Appl. Math. **206**, 73–80 (2016)
5. van Aardt, S.A., Burger, A.P., Dunbar, J.E., Frick, M., Llano, B., Thomassen, C., Zuazua, R.: Destroying longest cycles in graphs and digraphs. Discret. Appl. Math. **186**(Suppl. C), 251–259 (2015)
6. Fernandes, C., Gutiérrez, J.: Hitting all longest cycles in a graph. In: Anais do XXXVII congresso da sociedade brasileira de computação, pp. 87–90 (2017)
7. Balister, P., Győri, E., Lehel, J., Schelp, R.: Longest paths in circular arc graphs. Comb. Probab. Comput. **13**(3), 311–317 (2004)
8. Cerioli, M., Lima, P.: Intersection of longest paths in graph classes. Electron. Notes Discret. Math. **55**, 139–142 (2016)
9. Chen, F.: Nonempty intersection of longest paths in a graph with a small matching number. Czechoslov. Math. J. **65**(140), 545–553 (2015)
10. Chen, G., Ehrenmüller, J., Fernandes, C., Heise, C., Shan, S., Yang, P., Yates, A.: Nonempty intersection of longest paths in seriesparallel graphs. Discret. Math. **340**(3), 287–304 (2017)
11. de Rezende, S., Fernandes, C., Martin, D., Wakabayashi, Y.: Intersecting longest paths. Discret. Math. **313**, 1401–1408 (2013)
12. Golan, G., Shan, S.: Nonempty intersection of longest paths in $2K_2$-free graphs. https://arxiv.org/abs/1611.05967 (2016)
13. Klavžar, S., Petkovšek, M.: Graphs with nonempty intersection of longest paths. Ars Comb. **29**, 43–52 (1990)
14. Zamfirescu, T.: On longest paths and circuits in graphs. Math. Scand. **38**(2), 211–239 (1976)
15. Cerioli, M.R., Fernandes, C.G., Gómez, R., Gutiérrez, J., Lima, P.T.: Transversals of longest paths. Electron. Notes Discret. Math. **62**(Suppl. C), 135–140 (2017). IX Latin and American Algorithms, Graphs and Optimization, LAGOS 2017
16. Cerioli, M.R., Fernandes, C.G., Gómez, R., Gutiérrez, J., Lima, P.T.: Transversals of longest paths (2017). https://arxiv.org/abs/1712.07086
17. Chen, G., Faudree, R.J., Gould, R.J.: Intersections of longest cycles in k-connected graphs. J. Comb. Theory Ser. B **72**(1), 143–149 (1998)
18. Hippchen, T.: Intersections of longest paths and cycles. Ph.D. thesis, Georgia State University (2008)
19. Jendrol, S., Skupień, Z.: Exact numbers of longest cycles with empty intersection. Eur. J. Comb. **18**(5), 575–578 (1997)
20. Stewart, I.A., Thompson, B.: On the intersections of longest cycles in a graph. Exp. Math. **4**(1), 41–48 (1995)
21. Diestel, R.: Graph Theory. GTM, vol. 173, 4th edn. Springer, Heidelberg (2017)
22. Bodlaender, H.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**(1), 1–45 (1998)
23. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. J. Comb. Theory Ser. B **16**(1), 47–56 (1974)
24. Heinz, M.: Tree-decomposition: graph minor theory and algorithmic implications. Master's thesis, Technischen Universität Wien (2013)
25. Fomin, F., Thilikos, D.: New upper bounds on the decomposability of planar graphs. J. Graph Theory **51**(1), 53–81 (2006)
26. Alon, N., Seymour, P., Thomas, R.: A separator theorem for nonplanar graphs. J. Am. Math. Soc. **3**, 801–808 (1990)
27. Shabbir, A., Zamfirescu, C., Zamfirescu, T.: Intersecting longest paths and longest cycles: a survey. Electron. J. Graph Theory Appl. **1**, 56–76 (2013)

Majority Model on Random Regular Graphs

Bernd Gärtner and Ahad N. Zehmakan(✉)

Department of Computer Science, ETH Zurich, Zürich, Switzerland
{gaertner,abdolahad.noori}@inf.ethz.ch

Abstract. Consider a graph $G = (V, E)$ and an initial random coloring where each vertex $v \in V$ is blue with probability P_b and red otherwise, independently from all other vertices. In each round, all vertices simultaneously switch their color to the most frequent color in their neighborhood and in case of a tie, a vertex keeps its current color. The main goal of the present paper is to analyze the behavior of this basic and natural process on the random d-regular graph $\mathbb{G}_{n,d}$. It is shown that for $\epsilon > 0$, $P_b \leq 1/2 - \epsilon$ results in final complete occupancy by red in $\mathcal{O}(\log_d \log n)$ rounds with high probability, provided that $d \geq c/\epsilon^2$ for a sufficiently large constant c. We argue that the bound $\mathcal{O}(\log_d \log n)$ is asymptomatically tight. Furthermore, we show that with high probability, $\mathbb{G}_{n,d}$ is immune; i.e., the smallest dynamic monopoly is of linear size. A dynamic monopoly is a subset of vertices that can "take over" in the sense that a commonly chosen initial color eventually spreads throughout the whole graph, irrespective of the colors of other vertices. This answers an open question of Peleg [22].

Keywords: Majority model · Random regular graph
Bootstrap percolation · Density classification · Threshold behavior
Dynamic monopoly

1 Introduction

Consider a graph $G = (V, E)$ with an initial coloring where each vertex is red or blue. Each red/blue vertex could correspond to an infected/uninfected cell in a brain, a burning/non-burning tree in a forest, a positive/negative individual in a community regarding a reform proposal, or an informed/uninformed processor in a distributed system. Starting from an initial coloring, and in discrete-time rounds, all vertices synchronously update their current color based on a predefined rule as a function of the current coloring of their neighbors. By defining a sufficiently large updating rule, this process can model different basic dynamic phenomena, like infection spreading among cells, fire propagation in a forest, opinion forming regarding an election in a community, or information distribution among processors. As two simple examples, a tree starts burning if at least one of its neighbors is on fire, or a person adopts the most frequent opinion among his/her friends.

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 572–583, 2018.
https://doi.org/10.1007/978-3-319-77404-6_42

Researchers from a wide spectrum of fields, from biology to physics, and with various motivations, have extensively investigated the behavior of such processes. One of the most natural updating rules, whose different variants have attracted a substantial amount of attention, is the *majority rule* where a vertex updates its current color to the most frequent color in its neighborhood.

Here, one of the most studied variants is *majority bootstrap percolation* in which by starting from a random coloring, where each vertex is blue with probability P_b and red with probability $P_r = 1 - P_b$ independently, in each round a blue vertex switches to red if at least half of its neighbors are red, and a red vertex stays red forever. A considerable amount of effort has been put into the investigation and analysis of majority bootstrap percolation, both theoretically and experimentally, from results by Balogh et al. [3] to the recent paper by Stefánsson and Vallier [26]. Typical graphs are the d-dimensional lattice, the d-dimensional hypercube, the binomial random graph, and the random regular graph.

The main motivation behind majority bootstrap percolation is to model monotone dynamic processes like rumor spreading, where an informed individual will always stay informed of the rumor (corresponding to red color, say). However, it does not model non-monotone processes like opinion forming in a community, distributed fault-local mending, and diffusion of two competing technologies over a social network. For this, the following *majority model* is considered: we are given a graph $G = (V, E)$ and an initial random coloring, where each vertex is blue with probability P_b and red otherwise, independently of other vertices. In each round, all vertices simultaneously update their color to the most frequent color in their neighborhood; in case of a tie, a vertex keeps its current color. Since the majority model is a deterministic process on a finite state space, the process must reach a cycle of states after a finite number of rounds. The number of rounds that the process needs to reach the cycle is called the *consensus time* of the process.

Even though different aspects of the majority model like the consensus time and its threshold behavior have been studied both experimentally and theoretically (see Sect. 1.2 for more details), there is not much known about the behavior of the majority model on the random d-regular graph. Majority bootstrap percolation [4], rumor spreading [11,20], and flooding process [2] have been studied on random regular graphs, but this graph class is not easy to handle. Even though the behavior of majority bootstrap percolation on the random regular graph had been discussed in several prior works, it took almost two decades until Balogh and Pittel [4] could analyze the behavior of the process partially. They proved (under some limitations on the size of d) that there are two values P_1 and P_2 such that $P_r \ll P_1$ results in the coexistence of both colors and $P_2 \ll P_r$ results in a fully red configuration with high probability[1]; however, $P_1 \neq P_2$ which leaves a gap in the desired threshold behavior of the process. We shortly write $f(n) \ll g(n)$ for $f(n) = o(g(n))$.

[1] For an n-node graph $G = (V, E)$, we say an event happens with high probability (w.h.p.) if its probability is at least $1 - o(1)$ as a function of n. Notice we do not require the probability $1 - 1/n^c$, for a constant $c > 0$, as it is done in some contexts.

In the present paper, we prove that in the majority model on the random d-regular graph, and for $\epsilon > 0$, $P_b \leq 1/2 - \epsilon$ results in final complete occupancy by red color in $\mathcal{O}(\log_d \log n)$ rounds w.h.p. if $d \geq c/\epsilon^2$ for a sufficiently large constant c. In words, even a narrow majority takes over the whole graph extremely fast. We should point out that the result probably holds for $d \geq 3$, but our proof techniques do not yield this, since they require sufficient edge density in the underlying graph. We also show that the upper bound of $\mathcal{O}(\log_d \log n)$ is best possible.

A natural context of this result is the *density classification problem*; coming from the theory of cellular automata, this is the problem of finding an updating rule for a given graph G such that for any initial 2-coloring the process reaches a monochromatic configuration by the initial majority color. It turned out that the problem is hard in the sense that even for a cycle, there is no rule which can do the density classification task perfectly [19]. Our result shows that the majority rule does the density classification task acceptably for almost every d-regular graph with d sufficiently large (see Theorem 2 for the precise meaning of acceptably and sufficiently large).

It is an interesting (and currently unanswered question) which properties of a graph are chiefly responsible for the majority rule being able to almost classify density - or failing to do so. For example, we know that on a torus $T_{\sqrt{n},\sqrt{n}}$ (a $\sqrt{n} \times \sqrt{n}$ lattice with "wrap-around"), already a very small initial blue density of $P_b \gg 1/n^{1/4}$ prevents red color from taking over, w.h.p. [13,14]. A plausible explanation is that the torus $T_{\sqrt{n},\sqrt{n}}$ has a very low vertex/edge expansion in comparison to the random regular graph; however, we do not know whether expansion is indeed the right parameter to look at here.

As a concrete application of our main Theorem 2, we improve a result and answer an open question by Peleg [22]. Motivated by the problem of fault-local mending in distributed systems, he introduced the concept of immunity. An n-vertex graph G is (α, β)-*immune* if a set of $m \leq \beta n$ vertices with a common color can take over at most αm vertices in the next round in the majority model. Peleg proved that there exists a d-regular graph that is $(\frac{c_2 \log n}{d}, \beta)$-immune, for suitable constants $c_1, c_2, \beta > 0$ and $d \geq c_1$. He also showed that this result is tight up to a logarithmic factor. We close this logarithmic gap. Peleg also asked whether there exist regular graphs that are immune in the sense that no sub-linear size set of a common color can eventually take over the whole graph. We answer his question positively: with probability the random d-regular graph is immune.

The outline of the paper is as follows. After presenting basic definitions and prior research in Sects. 1.1 and 1.2, the behavior of the majority model on the random d-regular graph is analyzed in Sect. 2; the application to immunity is presented in Sect. 2.2.

1.1 Notation and Preliminaries

For a vertex v in graph $G = (V, E)$ the *neighborhood* of v is defined as $N(v) := \{u \in V : (v, u) \in E\}$. Furthermore for $u, v \in V$, let $d(u, v)$ denote the length of the shortest path between v, u in terms of the number of edges, which is called the *distance* between v and u (for a vertex v, we define $d(v, v) = 0$). For $v \in V$, $N_i(v) := \{u \in V : d(v, u) \leq i\}$ is the set of vertices in distance at most i from v.

A *generation* is a function $g : V \to \{b, r\}$ where b and r stand for blue and red, respectively. In addition to $g(v) = c$ for a vertex $v \in V$ and $c \in \{b, r\}$, we also write $g|_S = c$ for a set $S \subseteq V$ which means $\forall v \in S$, $g(v) = c$. For a graph $G = (V, E)$ and a random initial generation g_0, where $\forall v \in V$ $Pr[g_0(v) = b] = P_b$ and $Pr[g_0(v) = r] = P_r = 1 - P_b$ independently, assume $\forall i \geq 1$ and $v \in V$, $g_i(v)$ is equal to the color that occurs most frequently in v's neighborhood in g_{i-1}, and in case of a tie $g_i(v) = g_{i-1}(v)$. This model is called the *majority model.* Without loss of generality, we always assume that $P_b \leq P_r$.

The *random d-regular graph* $\mathbb{G}_{n,d}$ is the random graph with a uniform distribution over all d-regular graphs on n vertices, say $[n]$ (in this paper, we assume whenever talking about $\mathbb{G}_{n,d}$, dn is even). The definition of the random regular graph is conceptually simple, but it is not easy to use. However, there is an efficient way to generate $\mathbb{G}_{n,d}$ which is called the *configuration model* [5].

In the configuration model for $V = [n]$, which is to be the vertex set of the graph, we associate the d-element set $W_i = \{i\} \times [d] = \{(i, i') : 1 \leq i' \leq d\}$ to vertex $1 \leq i \leq n$. Let $W = [n] \times [d]$ be the union of W_is; then a *configuration* is a partition of W into $dn/2$ pairs. These pairs are called the edges of the configuration. The natural projection of the set W onto $V = [n]$ (ignoring the second coordinate) projects each configuration F to a multigraph $\pi(F)$ on V. Note that $\pi(F)$ might contain loops and multiple edges. Thus, $\pi(F)$ is not necessarily a simple graph. We define the *random d-regular multigraph* $\mathbb{G}^*_{n,d}$ to be the multigraph $\pi(F)$ obtained from a configuration F chosen uniformly at random among all configurations on W. Bender and Canfield [5] proved that if we consider $\mathbb{G}^*_{n,d}$ and condition on it being a simple graph, we obtain a random d-regular graph on V with uniform distribution over all such graphs. Furthermore, it is known [16] that if $Pr(\mathbb{G}^*_{n,d} \in A_n) \to 0$ as $n \to \infty$ then also $Pr(\mathbb{G}_{n,d} \in A_n) \to 0$, where A_n is a subset of d-regular multigraphs on V. This allows us to work with $\mathbb{G}^*_{n,d}$ instead of $\mathbb{G}_{n,d}$ itself in our context. To generate a random configuration, it suffices to define an arbitrary ordering on the elements of W and repeatedly match the first unmatched element in this order with another unmatched element uniformly at random. In Lemma 1, we utilize a slightly different construction from [7].

1.2 Prior Work

Even though a substantial amount of effort has been put into the study of a wide spectrum of the majority-based dynamic processes, our attention here is mostly devoted to the prior work concerning the majority model. However, let us briefly point out a couple of remarkable accomplishments regarding the majority bootstrap percolation, which is arguably the closest model to ours. Aizenmann and Lebowitz [1] proved that in the d-dimensional lattice there is a threshold value P_c so that $P_r \ll P_c$ and $P_c \ll P_r$ respectively result in the stable coexistence of both colors and fully red configuration with high probability Balogh and Bollobás [3] investigated the model on the d-dimensional hypercube and proved that the process has a phase transition with a sharp threshold. As discussed, the case of the random regular graph was also studied by Balogh and Pittel [4].

The majority model was introduced by Spitzer [25] in 1970. Afterwards, the model's behavior was investigated mostly by computer simulations (i.e., Monte-Carlo methods). These computer simulations (see for instance [8]) suggested that the model shows a threshold behavior on the two-dimensional torus $T_{\sqrt{n},\sqrt{n}}$. To address this observation, it was proven [14] that $P_b \ll n^{-1/4}$ and $P_b \gg n^{-1/4}$ respectively result in red monochromatic generation and the stable coexistence of both colors w.h.p. Furthermore Schonmann [24] proved in the biased variant of the majority model, where in case of a tie always red is chosen, and torus $T_{\sqrt{n},\sqrt{n}}$, for $1/\log n \ll P_r$ w.h.p. the process reaches fully red generation.

Since the updating rule is deterministic and there are $2^{|V|}$ possible colorings, the majority process must always reach a cycle of generations. Poljak and Turzík [23] showed that the number of rounds needed to reach the cycle (i.e., the consensus time) is $\mathcal{O}(|V|^2)$, and Goles and Olivos [15] proved the length of the cycle is always one or two. Frischknecht et al. [12] showed there exists graph $G = (V, E)$ which needs $\Omega(|V|^2/\log^2|V|)$ rounds to stabilize for some initial coloring in the majority model, which thus leaves only a poly-logarithmic gap. Kasser et al. [17] studied a decision variant of the problem; they proved for a given graph $G = (V, E)$ and an integer k, it is NP-complete to decide whether there exists an initial coloring for which the consensus time is at least k.

Kempe et al. [18], motivated from viral marketing, and independently Peleg [21], motivated from fault-local mending in distributed systems, introduced the concept of dynamic monopoly, a subset of vertices that can take over the whole graph. Afterwards, lots of studies regarding the size of dynamic monopolies and their behavior have been done. To name a few, even though it was conjectured [21] that the size of the smallest dynamic monopoly in the majority model is $\Omega(\sqrt{|V|})$ for a graph $G = (V, E)$, Berger [6], surprisingly, proved there exist graphs with dynamic monopolies of constant size. Furthermore, Flocchini et al. [10] studied the size of the smallest dynamic monopoly in the two dimensional torus. For more related results regarding dynamic monopolies, the interested reader is referred to a more recent work by Peleg [22].

2 Majority Model on Random Regular Graphs

The three special cases of $d = 0, 1, 2$ are exceptions to many properties of the random d-regular graph $\mathbb{G}_{n,d}$. For instance, $\mathbb{G}_{n,d}$ is d-connected for $d \geq 3$, but disconnected for $d \leq 2$ w.h.p. [16]. In the majority model also these three special cases show a different sort of behavior, which intuitively comes from their disconnectivity. We shortly discuss these cases following two purposes. Firstly, their threshold behavior sounds interesting by its own sake. Secondly, as a warm-up it probably helps the reader to have a better understanding of the majority model before going through our main results and proof techniques concerning the density classification in Sect. 2.1 and dynamic monopolies in Sect. 2.2.

A 0-regular graph is an empty graph with n vertices, and a 1-regular graph is the same as a perfect matching. We argue that in both cases $P_b \ll 1/n$ results in red monochromatic generation and $P_b \gg 1/n$ results in the coexistence of both colors w.h.p. (recall we assume $P_b \leq P_r$). Let random variable X denote the number of blue vertices in the initial generation. $\mathbb{E}[X] = nP_b = o(1)$ for $P_b \ll 1/n$, and by Markov's inequality [9] w.h.p. $g_0|_V = r$. If $1/n \ll P_b$, then $\mathbb{E}[X] = \omega(1)$. Since X is the sum of n independent Bernoulli random variables, Chernoff bound [9] implies that w.h.p. there exists a blue vertex in the initial generation, which guarantees the survival of blue color in both cases.

We show the random 2-regular graph $\mathbb{G}_{n,2}$ also has a phase transition, but at $1/\sqrt{n}$ instead of $1/n$. Actually more strongly, we prove that for any n-vertex 2-regular graph, $P_b \ll 1/\sqrt{n}$ and $1/\sqrt{n} \ll P_b$ w.h.p. result in fully red generation and the stable coexistence of both colors, respectively. Notice a 2-regular graph is the union of cycles of length at least 3.

Theorem 1. *In the majority model and an n-vertex 2-regular graph $G = (V, E)$, $P_b \ll 1/\sqrt{n}$ results in red monochromatic generation and $1/\sqrt{n} \ll P_b$ outputs the stable coexistence of both colors w.h.p.*

Proof. In a generation g, define a blue (red) edge to be an edge whose both endpoints are blue (red). Consider an arbitrary edge set $E' \subset E$ which contains linearly many disjoint edges (a maximum matching, say), and let random variable X_1 denote the number of blue edges of E' in g_0. For $1/\sqrt{n} \ll P_b$, $\mathbb{E}[X_1] = \omega(1)$; thus, by Chernoff bound there is a blue edge in g_0 w.h.p. which guarantees the survival of blue color.

If $P_b \ll 1/\sqrt{n}$, then $\mathbb{E}[X_2] = o(1)$, where the random variable X_2 denotes the number of blue edges in g_0, which by Markov's inequality implies there is no blue edge in g_0 w.h.p. If in a cycle there is no blue edge and there is at least a red edge, then the cycle gets red monochromatic after at most $n/2$ rounds because the red edge grows from both sides in each round until it covers the whole cycle. Thus, it only remains to show that each cycle contains a red edge w.h.p. An odd cycle always contains a monochromatic edge (red in our case). Then, let X_3 denote the number of even cycles which contain no monochromatic edge; i.e., the vertices are red and blue one by one. Define n_i to be the number of cycles of length i. We have

$$\mathbb{E}[X_3] \leq \sum_{2\leq i\leq \lfloor n/2 \rfloor} n_{2i} \cdot 2P_b^i(1-P_b)^i \leq 2P_b^2 \sum_{2\leq i\leq \lfloor n/2 \rfloor} n_{2i} = o(1)$$

where we used $P_b \ll 1/\sqrt{n}$ and the fact that there are at most linearly many cycles. Therefore, w.h.p. there is no cycle without a red edge. □

2.1 Density Classification

In this section, it is shown that in the d-regular random graph $\mathbb{G}_{n,d}$ and the majority model, $P_b \leq 1/2 - \epsilon$, for $\epsilon > 0$, results in fully red generation in $\mathcal{O}(\log_d \log n)$ rounds w.h.p. provided that $d \geq c/\epsilon^2$ for a sufficiently large constant c. To prove that, first we need to provide Lemmas 1 and 2 as the ingredients, which are also interesting and important by their own sake. Specifically, the results in Sect. 2.2 concerning dynamic monopolies and immunity are built on Lemma 2.

We say the k-neighborhood of a vertex v in a graph G is a tree if the induced subgraph by vertex set $N_k(v)$ is a tree. Roughly speaking, Lemma 1 explains that for small d and k the expected number of vertices whose k-neighborhood is not a tree in $\mathbb{G}_{n,d}$ is small. This local tree-like structure turns out to be very useful in bounding the consensus time of the process.

Lemma 1. *In $\mathbb{G}_{n,d}$, the expected number of vertices whose k-neighborhood is not a tree is at most $4d^{2k}$.*

Proof. Firstly, we assume $k < \log_d(n/2)$ because otherwise the statement is clearly true. Furthermore as discussed in Sect. 1.1, we work with the random d-regular multigraph $\mathbb{G}^*_{n,d}$ instead of the random d-regular graph $\mathbb{G}_{n,d}$, on vertex set $V = [n]$. We generate a uniformly at random configuration by partitioning $W = \bigcup_{1 \le i \le n} W_i$ into $dn/2$ pairs as follows, where the d-element set $W_i = \{i\} \times [d]$ corresponds to the vertex $1 \le i \le n$. In step 1, we start from an arbitrary class (we utilize the terms of d-element set and class interchangeably), say W_1, and match its elements one by one based on an arbitrary predefined order with an unmatched element (from W_1 or other classes) uniformly at random. We say a class has been *reached* in step $j \ge 1$ if for the first time in step j one of its elements has been matched. In step $j \ge 2$, we match the unmatched elements of the classes reached in step $j-1$ one by one based on a predefined ordering, say lexicographical order, with unmatched elements uniformly at random. It is possible in some step, no new class is reached. In this case, if all elements are matched, the process is over; otherwise we continue the process from one of the unreached classes, say the one with the smallest index.[2]

Assume in step $j \ge 1$ we match element x with an element y, chosen uniformly at random among all yet unmatched elements. We say xy is a *cycle-maker* if y is not the first element matched in its class. The probability that an edge selected in the j-th step is a cycle-maker is at most $d^j/(n-d^j)$. Thus, the probability that there is a cycle-maker edge in the first k steps is at most $2d^k \cdot \max_{1\le j\le k} \frac{d^j}{n-d^j}$ which is smaller than $\frac{2d^{2k}}{n-d^k}$. Let X denote the number of vertices whose k-neighborhood is not a tree. Then, we have $\mathbb{E}[X] \le (2nd^{2k})/(n-d^k)$ which is smaller than $4d^{2k}$ for $k < \log_d(n/2)$ because $n - d^k > n - d^{\log_d(n/2)} = n/2$. □

Corollary 1. *In $\mathbb{G}_{n,d}$, the number of vertices whose $(c' \log_d \log_2 n)$-neighborhood is not a tree is at most $\log_2^{2c'+1} n$ w.h.p., for constant $c' > 0$.*

Proof. Let X denote the number of vertices whose $(c' \log_d \log_2 n)$-neighborhood is not a tree. By Lemma 1, $\mathbb{E}[X] \le 4d^{2c' \log_d \log_2 n} = 4 \log_2^{2c'} n$. By Markov's inequality $Pr[X \ge \log_2^{2c'+1} n] \le 4/\log_2 n = o(1)$. □

In a graph $G = (V, E)$ for two (not necessarily disjoint) vertex sets S and S', we say that S *controls* S' if S being monochromatic in some generation implies S' being monochromatic (of the same color) in the next generation in the majority

[2] For a more formal description of the construction, please see [7], and notice since the second element always is chosen randomly, the generated configuration is random.

model, irrespective of the colors of other vertices. Clearly in $\mathbb{G}_{n,d}$, S controls S' implies that for every $v \in S'$ at least $\lceil d/2 \rceil$ of its neighbors are in S.

Lemma 2. *In $\mathbb{G}_{n,d}$ on vertex set $V = [n]$ with $d \geq c_1$, w.h.p. there do not exist two vertex sets S, S' such that S controls S', $|S| \leq \frac{n}{c''}$, $|S'| = \lceil \frac{10|S|}{d} \rceil$, where c_1, c'' are sufficiently large constants.*

This immediately implies that in $\mathbb{G}_{n,d}$ and the majority model, less than $\frac{n}{c''}$ blue (red) vertices will die out in $\mathcal{O}(\log_d n)$ rounds w.h.p.

Proof. We fix two sets S, S' of the given sizes s and s'. We show that the probability for S controlling S' is so small that a union bound over all pairs (S, S') yields the desired high probability result. We equivalently work in $\mathbb{G}^*_{n,d}$ the relevant "initial" part of which we generate as follows: we iterate through the pairs in $S' \times [d]$ in some fixed order and match each yet unmatched pair with a random unmatched pair in $V \times [d]$. In order for S to control S', at least $\lceil d/2 \rceil$ of the d pairs (v, i) must get matched with pairs in $S \times [d]$, for every $v \in S'$. Overall, at least $\lceil d/2 \rceil s'$ of the ds' pairs $S' \times [d]$ get matched with pairs in $S \times [d]$. Such a match is established only when the randomly chosen partner happens to be in $(S \cup S') \times [d]$, and this may actually yield two of the required $\lceil d/2 \rceil s'$ pairs. Hence, for S to control S', at least $\ell := \lceil d/2 \rceil s'/2$ of the $L := ds'$ iterations must be *active*, meaning that they match a yet unmatched pair with a pair in $(S \cup S') \times [d]$. For a bit vector $\mathbf{b}$ of length at most L, let $A(\mathbf{b})$ denote the event that iteration i is active for exactly the indices where $b_i = 1$. Then $Pr[A(b_1, b_2 \ldots, b_L)] = \prod_{i=1}^{L} Pr[\text{iteration } i \text{ is active if } b_i = 1 | A(b_1, \ldots, b_{i-1})]$ (the right-hand side is a telescoping product). Now, irrespective of $b_1, \ldots, b_{i-1}$, an iteration is active with probability at most $d(s + s')/(nd - 2ds') = (s + s')/(n - 2s') \leq 2s/n$. Hence, for a vector $\mathbf{b}$ with at least ℓ ones, $Pr[A(b_1, b_2 \ldots, b_L)] \leq (2s/n)^{\ell}$. As there are at most 2^L such vectors, the probability that at least ℓ iterations are active is at most $2^L (2s/n)^{\ell} \leq 2^{10s} (2s/n)^{\frac{5}{2}s}$. Hence by a union bound, the probability P that there exist such sets S and S' in a random configuration is at most $\sum_{s=1}^{\frac{n}{c''}} \binom{n}{s} \binom{n}{\lceil \frac{10s}{d} \rceil} 2^{10s} (\frac{2s}{n})^{\frac{5}{2}s}$. Since $d \geq c_1$ for a large constant c_1, $\binom{n}{\lceil \frac{10s}{d} \rceil} \leq \binom{n}{s}$; thus, applying Stirling's approximation [9] (i.e., $\binom{n}{k} \leq (ne/k)^k$) yields $P \leq \sum_{s=1}^{\frac{n}{c''}} (\frac{ne}{s})^{2s} 2^{10s} (\frac{2s}{n})^{\frac{5}{2}s}$. Furthermore, since $e^{2s} \cdot 2^{10s} \cdot 2^{\frac{5}{2}s} \leq (c'')^{s/4} \leq (n/s)^{s/4}$ for sufficiently large c'', we have $P \leq \sum_{s=1}^{\frac{n}{c''}} (\frac{n}{s})^{2s} (\frac{s}{n})^{\frac{9}{4}s} = \sum_{s=1}^{\frac{n}{c''}} (\frac{s}{n})^{\frac{s}{4}} = o(1)$. □

As will be discussed in the proof of Theorem 2, for the majority model on $\mathbb{G}_{n,d}$ with $P_b = 1/2 - \epsilon$ and $d \geq c/\epsilon^2$ there is a simple argument which shows that the expected density of the blue vertices drops from $1/2 - \epsilon$ in g_0 to an arbitrarily small constant in g_1 if we select the constant c sufficiently large. Therefore, one might want to apply Lemma 2 to show the process w.h.p. gets red monochromatic in $\mathcal{O}(\log_d n)$ rounds. However, in Theorem 2 we show actually $\mathcal{O}(\log_d \log n)$ rounds suffice to get red monochromatic w.h.p. To prove that, we need the tree structure argued in Lemma 1.

Theorem 2. *In the majority model and $\mathbb{G}_{n,d}$, by starting from $P_b \leq 1/2 - \epsilon$, for $\epsilon > 0$, the process gets red monochromatic in $\mathcal{O}(\log_d \log n)$ rounds w.h.p. provided that $d \geq c/\epsilon^2$ for sufficiently large constant c.*

Proof. We say a vertex is in the j-th level of a rooted tree if its distance to the root is j. Now, consider a tree T rooted at vertex v and of height k so that except the vertices in the k-th level (i.e., leaves), all vertices are of degree d. We consider the following process on T, which we call the *propagation process*, where in the initial configuration all the internal vertices are *inactive* and each leaf is blue with probability P_b and red with probability $1 - P_b$ independently. Assume in each round an inactive vertex whose children are colored adopts color blue if at least $\lfloor (d-1)/2 \rfloor$ of its children are blue and red otherwise. Clearly after k rounds, the root (vertex v) is colored with blue or red. Let P_i for $0 \leq i \leq k$ denote the probability that a vertex in the $(k-i)$-th level is blue after round i; specifically, P_k is the probability that vertex v is blue at the end of the process. More accurately, $P_0 = P_b$ and for $1 \leq i \leq k$ $P_i = \sum_{j=\lfloor (d-1)/2 \rfloor}^{d-1} \binom{d-1}{j} P_{i-1}^j (1 - P_{i-1})^{d-1-j}$.

Now, let us get back to the majority model and the random d-regular graph $\mathbb{G}_{n,d}$. Consider a vertex v so that the induced subgraph by $N_k(v)$ is a tree T. Clearly in T, except the vertices in the k-th level, all vertices are of degree d. Now, we claim the probability that vertex v is blue in generation g_k in the majority model is at most P_k, which is equivalent to the probability that the root of T is blue in the propagation process after k rounds, with the same P_b. This is true because by starting with the same coloring for the leaves of T (the vertices in distance k from root v) if in the k-th round in the propagation process the root is red, it is also red in the majority model and generation g_k, irrespective of the colors of other vertices. For $k = 1$ this is trivially true. Now, we do the induction on k; if the root is red in the propagation process after k-th round, it means that there exist less than $\lfloor (d-1)/2 \rfloor$ blue vertices among root's children in round $k-1$ which implies by the induction hypothesis there are less than $\lfloor (d-1)/2 \rfloor$ blue vertices in v's neighborhood in g_{k-1} in the majority model; then $g_k(v) = r$.

So far, we showed the probability of being blue in g_k for a vertex, whose k-neighborhood is a tree, is at most P_k with $P_0 = P_b$. Now, we upper-bound the probability P_k. Without loss of generality, assume d is odd, and suppose $d' = d - 1$. First, let us bound P_1; clearly, $P_1 \leq \sum_{j=d'/2}^{d'} \binom{d'}{j} (1/2 - \epsilon)^j (1/2 + \epsilon)^{d'-j}$ which is smaller than

$$(1/2 - \epsilon)^{d'/2} (1/2 + \epsilon)^{d'/2} \sum_{j=d'/2}^{d'} \binom{d'}{j} \leq (1/4 - \epsilon^2)^{d'/2} 2^{d'} = (1 - 4\epsilon^2)^{d'/2}.$$

By applying the estimate $1 - x \leq e^{-x}$, we have $P_1 \leq e^{-2d'\epsilon^2}$. For $d \geq c_1' \log n$, where c_1' is a large constant, clearly $P_1 \leq 1/n^2$ which implies the expected number of blue vertices in g_1 is at most $1/n$; i.e., the process gets red monochromatic in one round w.h.p. Thus, it only remains to discuss the case of $d \leq c_1' \log n$ for an arbitrarily large constant c_1'; in this case, since $P_1 \leq e^{-2d'\epsilon^2}$, selecting suitable constant c, for $d \geq c/\epsilon^2$, results in $P_1 \leq 1/16$. Now, we show $P_i \leq P_{i-1}^{d'/4}$ for $P_{i-1} \leq 1/16$, which yields $P_k \leq 1/n^2$ for $k = c' \log_d \log_2 n$ by selecting constant c' large enough. We know $P_i \leq P_{i-1}^{d'/2} \sum_{j=d'/2}^{d'} \binom{d'}{j} \leq P_{i-1}^{d'/2} 2^{d'}$. Thus, by utilizing $P_{i-1} \leq 1/16$, one has $P_i \leq P_{i-1}^{d'/4}$. Now, let random variable X_1 (X_2) denote the number of vertices whose k-neighborhood for $k = c' \log_d \log_2 n$, is (not) a tree and are blue in g_k. We know $\mathbb{E}[X_1] \leq nP_k \leq 1/n$, which implies $X_1 = 0$ w.h.p. by Markov's inequality. Furthermore, by using Corollary 1 w.h.p. $X_2 \leq \log_2^{2c'+1} n$. Hence, w.h.p. the number of blue vertices in g_k is upper bounded by $\log_2^{2c'+1} n$. However based on Lemma 2, poly-logarithmically many blue vertices die out in $\mathcal{O}(\log_d \log n)$ rounds w.h.p. which finishes the proof. □

Now, we argue that the bound of $\mathcal{O}(\log_d \log n)$ is tight. We prove in $\mathbb{G}_{n,d}$ and for a constant small initial density P_b, say $P_b = 1/4$, after $k' = \frac{\log_d \log_2 n}{2}$ rounds w.h.p. there exist some blue vertices. We claim in $\mathbb{G}_{n,d}$ there are $\sqrt{n}$ vertices, say $u_1, \cdots, u_{\sqrt{n}}$, whose k'-neighborhood is pairwise disjoint. Define indicator random variable x_i to be 1 if $g_0|_{N_{k'}(u_i)} = b$. Clearly, $Pr[x_i = 1] \geq (1/4)^{2d^{k'}} = 1/2^{4\sqrt{\log_2 n}}$. Let $X = \sum_{i=1}^{\sqrt{n}} x_i$; then $\mathbb{E}[X] \geq \sqrt{n}/2^{4\sqrt{\log_2 n}} = \omega(1)$. By using Chernoff bound, there exists a vertex v so that $g_0|_{N_{k'}(v)} = b$, which implies $g_{k'}(v) = b$. Now, we prove that there exist $\sqrt{n}$ vertices whose k'-neighborhood is pairwise disjoint in every d-regular graph. For a d-regular graph G by starting from the state that all vertices are *unmarked*, recursively we choose an arbitrary unmarked vertex u and add u to set U, which is initially empty, and mark all vertices in $N_{2k'}(u)$. Clearly, the vertices in set U have our required disjointness property, and set U will be of size larger than $\sqrt{n}$ at the end because in each step we mark at most poly-logarithmically many vertices while we start with linearly many unmarked vertices.

2.2 Dynamic Monopoly and Immunity

In distributed systems, the resolution of inconsistencies by the majority rule is a common tool; the idea is to keep redundant copies of data and perform the majority rule to overcome the damage caused by failures. Motivated from this application, one might be interested in the networks in which no small subset of malicious/failed processors can take over a large part of the network. To address this issue, Peleg [22] suggested the concept of immunity. An n-vertex graph G is (α, β)-*immune* if a set of $m \leq \beta n$ vertices with a common color can take over at most αm vertices in the next round. One is interested in graphs which are (α, β)-immune for constant β and small α because roughly speaking these graphs are acceptably tolerant of malicious/failed vertices (processors). Peleg [22] proved the following theorem regarding the existence of such graphs.

Theorem 3. [22] *There exist constants $c_1, c_2, \beta > 0$ such that for every $d \geq c_1$, there exists a d-regular graph G which is $(\frac{c_2 \log n}{d}, \beta)$-immune.*

Peleg also argued that this result is tight up to a logarithmic factor, meaning there is a constant $c_2 > 0$ such that for any constant $\beta > 0$, there exist no $(\frac{c_2}{d}, \beta)$-immune d-regular graph. Now as an immediate implication of Lemma 2, we present Corollary 2 which improves upon Peleg's results by removing the extra logarithmic term. Hence, this result is tight up to a constant.

Corollary 2. *There exist constants $c_1, c_2, \beta > 0$ such that for every $d \geq c_1$, there exists a d-regular graph G which is $(\frac{c_2}{d}, \beta)$-immune.*

Furthermore, we say a graph is *immune* if the smallest dynamic monopoly is of linear size, in terms of the number of vertices. We recall that for a graph $G = (V, E)$, a set $D \subseteq V$ is called a dynamic monopoly whenever the following holds: if in the initial generation all vertices of D are blue (red) then the process reaches the blue (red) monochromatic generation, irrespective of the colors of other vertices. As an open problem, Peleg [22] asked that whether there exist regular immune graphs. The existence of immune d-regular graphs for $d \gg \log n$ is straightforward from Theorem 3, but, the question is unanswered for small d, while one is more interested in sparse immune regular graphs from a practical, or even a theoretical, perspective. Again, as an immediate result of Lemma 2, we have Corollary 3 which actually represents a stronger statement.

Corollary 3. *$\mathbb{G}_{n,d}$ with $d \geq c_1$ is immune w.h.p. for large constant c_1.*

3 Conclusion

We claim our techniques can be applied to analyze the behavior of the majority model on the binomial random graph $\mathbb{G}_{n,p}$. For $p \gg \log n/n$, one can show $P_b \leq 1/2 - \epsilon$ results in fully blue generation in one round w.h.p. by using the argument regarding the case of $d \geq c_1' \log n$ in the proof of Theorem 2. For $p \ll \log n/n$ the graph contains a red and a blue isolated vertex in g_0 w.h.p. for a fixed $P_b > 0$ which result in the coexistence of both colors.

Exploring the relation between the behavior of the majority model and the expansion level of the underlying graph can be a prospective research direction. Specifically, it would be interesting to prove that graphs with some certain level of expansion have a density classification behavior similar to $\mathbb{G}_{n,d}$.

Acknowledgments. The authors would like to thank Mohsen Ghaffari for several stimulating conversations and Jozsef Balogh and Nick Wormald for referring to some relevant prior results.

References

1. Aizenman, M., Lebowitz, J.L.: Metastability effects in bootstrap percolation. J. Phys. A: Math. Gen. **21**(19), 3801 (1988)
2. Amini, H., Draief, M., Lelarge, M.: Flooding in weighted sparse random graphs. SIAM J. Discrete Math. **27**(1), 1–26 (2013)
3. Balogh, J., Bollobás, B., Morris, R.: Majority bootstrap percolation on the hypercube. Comb. Probab. Comput. **18**(1–2), 17–51 (2009)
4. Balogh, J., Pittel, B.G.: Bootstrap percolation on the random regular graph. Random Struct. Algorithms **30**(1–2), 257–286 (2007)
5. Bender, E.A., Canfield, E.R.: The asymptotic number of labeled graphs with given degree sequences. J. Comb. Theory Ser. A **24**(3), 296–307 (1978)

6. Berger, E.: Dynamic monopolies of constant size. J. Comb. Theory Ser. B **83**(2), 191–200 (2001)
7. Bollobás, B., Fernandez de la Vega, W.: The diameter of random regular graphs. Combinatorica **2**(2), 125–134 (1982)
8. de Oliveira, M.J.: Isotropic majority-vote model on a square lattice. J. Stat. Phys. **66**(1), 273–281 (1992)
9. Feller, W.: An Introduction to Probability Theory and Its Applications: Volume I, vol. 3. Wiley, New York (1968)
10. Flocchini, P., Lodi, E., Luccio, F., Pagli, L., Santoro, N.: Dynamic monopolies in tori. Discrete Appl. Math. **137**(2), 197–212 (2004)
11. Fountoulakis, N., Panagiotou, K.: Rumor spreading on random regular graphs and expanders. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX/RANDOM-2010. LNCS, vol. 6302, pp. 560–573. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15369-3_42
12. Frischknecht, S., Keller, B., Wattenhofer, R.: Convergence in (social) influence networks. In: Afek, Y. (ed.) DISC 2013. LNCS, vol. 8205, pp. 433–446. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41527-2_30
13. Gärtner, B., Zehmakan, A.N.: (Biased) majority rule cellular automata. arXiv preprint arXiv:1711.10920 (2017)
14. Gärtner, B., Zehmakan, A.N.: Color war: cellular automata with majority-rule. In: Drewes, F., Martín-Vide, C., Truthe, B. (eds.) LATA 2017. LNCS, vol. 10168, pp. 393–404. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53733-7_29
15. Goles, E., Olivos, J.: Comportement périodique des fonctions à seuil binaires et applications. Discrete Appl. Math. **3**(2), 93–105 (1981)
16. Janson, S., Luczak, T., Rucinski, A.: Random Graphs, vol. 45. Wiley, Hoboken (2011)
17. Kaaser, D., Mallmann-Trenn, F., Natale, E.: On the voting time of the deterministic majority process. arXiv preprint arXiv:1508.03519 (2015)
18. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 137–146. ACM (2003)
19. Land, M., Belew, R.K.: No perfect two-state cellular automata for density classification exists. Phys. Rev. Lett. **74**(25), 5148 (1995)
20. Mourrat, J.-C., Valesin, D., et al.: Phase transition of the contact process on random regular graphs. Electron. J. Probab. **21** (2016)
21. Peleg, D.: Local majority voting, small coalitions and controlling monopolies in graphs: a review. In: Proceedings of 3rd Colloquium on Structural Information and Communication Complexity, pp. 152–169 (1997)
22. Peleg, D.: Immunity against local influence. In: Dershowitz, N., Nissan, E. (eds.) Language, Culture, Computation. Computing - Theory and Technology. LNCS, vol. 8001, pp. 168–179. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-45321-2_8
23. Poljak, S., Turzík, D.: On pre-periods of discrete influence systems. Discrete Appl. Math. **13**(1), 33–39 (1986)
24. Schonmann, R.H.: Finite size scaling behavior of a biased majority rule cellular automaton. Phys. A: Stat. Mech. Appl. **167**(3), 619–627 (1990)
25. Spitzer, F.: Interaction of Markov processes. Adv. Math. **5**(2), 246–290 (1970)
26. Stefánsson, S.Ö., Vallier, T.: Majority bootstrap percolation on the random graph G(n, p). arXiv preprint arXiv:1503.07029 (2015)

Property Testing for Point Sets on the Plane

Jie Han, Yoshiharu Kohayakawa, Marcelo Tadeu Sales, and Henrique Stagni(✉)

Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão 1010, São Paulo 05508-090, Brazil
{jhan,yoshi,mtsales,stagni}@ime.usp.br

Abstract. A *configuration* is a point set on the plane, with no three points collinear. Given three non-collinear points p, q and $r \in \mathbb{R}^2$, let $\chi(p,q,r) \in \{-1,1\}$, with $\chi(p,q,r) = 1$ if and only if, when we traverse the circle defined by those points in the counterclockwise direction, we encounter the points in the cyclic order $p,q,r,p,q,r,\ldots$. For simplicity, extend χ by setting $\chi(p,q,r) = 0$ if p, q and r are not pairwise distinct. Two configurations A and $B \subset \mathbb{R}^2$ are said to have the same *order type* if there is a bijection $\iota : A \to B$ such that $\chi(p,q,r) = \chi(\iota(p),\iota(q),\iota(r))$ for all $(p,q,r) \in A^3$. We say that a configuration C contains a *copy* of a configuration A if there is $B \subset C$ with A and B of the same order type. Given a configuration F, let $\mathrm{Forb}(F)$ be the set of configurations that do not contain a copy of F. The *distance* between two configurations A and B with $|A| = |B| = n$ is given by

$$\mathrm{dist}(A,B) = \min_{\iota} \frac{1}{2n^3} \sum_{(p,q,r)\in A^3} |\chi(p,q,r) - \chi(\iota(p),\iota(q),\iota(r))|,$$

where the minimum is taken over all bijections $\iota : A \to B$. Roughly speaking, we prove the following property testing result: *being free of a given configuration is efficiently testable*. Our result also holds in the general case of *hereditary properties* $\mathcal{P} = \mathrm{Forb}(\mathcal{F})$, defined by possibly infinite families $\mathcal{F}$ of forbidden configurations. Our results complement previous results by Czumaj, Sohler and Ziegler and others, in that we use a different notion of distance between configurations. Our proofs are heavily inspired on recent work of Fox and Wei on testing permutations and also make use of the regularity lemma for semi-algebraic hypergraphs of Fox, Pach and Suk. An extremal function involving order types, which may be of independent interest, plays an important rôle in our arguments.

1 Introduction

The *decision problem* associated with a property $\mathcal{P}$ consists in, given an instance I, distinguishing between the cases $I \in \mathcal{P}$ and $I \notin \mathcal{P}$. In *property testing*, as introduced by Rubinfeld and Sudan [16], one considers a relaxed version of this problem: one requires that one should distinguish, with a randomized algorithm, between the case in which $I \in \mathcal{P}$ and the case in which I is 'far' from $\mathcal{P}$. In return, one is interested in doing this very fast: one usually requires sublinear algorithms or algorithms with small 'query complexity', that is, algorithms that examine the given instance in a limited number of random spots, with 'limited' often meaning that the number of spot-checks should be independent of the size of the instance. In property testing, we also require a notion of *distance* for the objects involved (recall that we wish to distinguish objects in the given property $\mathcal{P}$ and 'far' from $\mathcal{P}$). In fact, the choice of distance may lead to quite different results. Property testing has become a very lively area of research, with important developments on the testing of graph and hypergraph properties, algebraic properties, properties of functions, properties of distributions, and geometric properties. The reader is referred to [11] for an overview of this area.

Our focus in this paper is on geometric problems. Our perspective complements the approaches in [4,5,8], in that we use a different notion of distance for the geometric objects in question. We consider problems involving finite point sets in general position, which we call *configurations*, and, for simplicity, we restrict ourselves to the 2-dimensional case. Furthermore, we work in the category of *order types* [13] (that is, realizable oriented matroids [1] or chirotopes [2]; see also [15]). A basic problem that can be cast in this framework is that of testing whether a point set S is in *convex position*, that is, whether every element of S is a *vertex* of the convex hull $\operatorname{conv}(S)$ of S (that is, an extreme point of $\operatorname{conv}(S)$). This problem was investigated in the seminal papers [4,5,8], with different notions of distance (see also [3] for a recent study of the high-dimensional case). Our point of view makes it natural to consider the property of being convex as a certain *hereditary property* of order types (a property that is closed under taking subsets), and it also suggests an arguably 'natural' notion of distance.

We present two results on testing general hereditary properties of configurations, Theorems 1 and 2, and we present a result on a certain extremal function k_F^* involving configurations, Theorem 3, that implies that certain numerical parameters appearing in Theorems 1 and 2 can be much improved for certain hereditary properties (namely, those defined as the configurations that avoid a fixed 'forbidden' configuration F that satisfies the so-called Erdős–Hajnal property (see Corollary 1 in Sect. 4)).

Our general approach is heavily inspired on recent work of Fox and Wei [9] addressing property testing problems for permutations under different notions of distance. Another tool that is useful in our context is the regularity lemma for semi-algebraic hypergraphs of Fox et al. [10].

This paper is organized as follows. In Sect. 2, we state Theorems 1 and 2, after introducing the required definitions. The extremal function k_F^* mentioned above is introduced in Sect. 3 and Theorem 3 is stated and proved in Sect. 4. Section 5 contains the proofs of Theorems 1 and 2.

2 Formal Set-Up and Main Results

We define a *configuration* of points as a finite set $C \subset \mathbb{R}^2$ with no three of them collinear. The *orientation* $\chi(p_1, p_2, p_3)$ of a triple of points (p_1, p_2, p_3) in the plane is given by

$$\chi(p_1,p_2,p_3) = \operatorname{sign}\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix}, \tag{1}$$

where the x_i and y_i are the coordinates of the p_i $(1 \le i \le 3)$. Thus $\chi(p_1,p_2,p_3) \in \{-1,0,1\}$. We say that two configurations C and $C' \subset \mathbb{R}^2$ are *isomorphic* (denoted by $C \simeq C'$) or that they have the same *order-type* if there is a bijection $\iota : C \to C'$ satisfying $\chi(u,v,w) = \chi(\iota(u),\iota(v),\iota(w))$ for all $(u,v,w) \in C^3$. We say that a configuration C contains a *copy* of a configuration F if there is $F' \subset C$ such that F' is isomorphic to F. In such cases, when there is no danger of confusion, we simply write $F \subset C$.

For every (possibly infinite) family $\mathcal{F}$ of configurations, we are interested in the property Forb($\mathcal{F}$) of all $\mathcal{F}$-*free* configurations, i.e., all configurations having no copy of members $F \in \mathcal{F}$. If $\mathcal{F} = \{F\}$, we denote the property of all F-*free* configurations simply by Forb(F).

It is easy to see that every hereditary property $\mathcal{P}$ can be written as $\mathcal{P} = \text{Forb}(\mathcal{F})$ for some $\mathcal{F}$ — in fact, one can take $\mathcal{F}$ as the set of all (*minimal*) configurations not in $\mathcal{P}$. As an example, let Conv be the set of all point configurations in convex position, i.e., the set of all configurations $C = \{v_i\}_{i=1}^n$ for which $\chi(v_i, v_{(i+1) \bmod n}, v) = 1$ for every $v \notin \{v_i, v_{(i+1) \bmod n}\}$. Then Conv can be written as Conv $= \text{Forb}(N_4)$, where N_4 is the unique four vertex configuration not in Conv shown in Fig. 2.

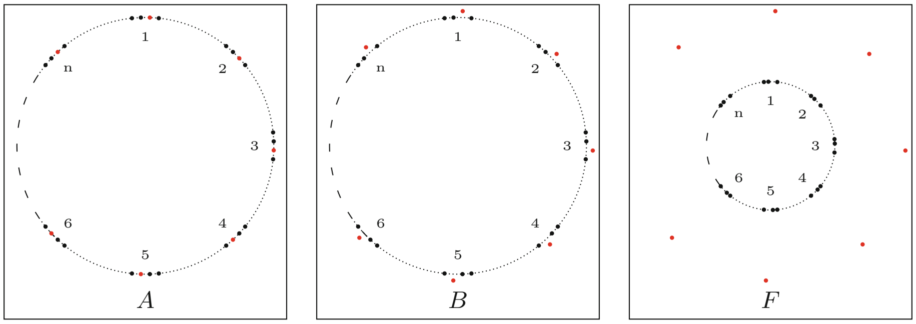


Fig. 1. Configuration B is a perturbation of a configuration A of $4n$ points in convex position. (Color figure online)

Czumaj et al. [5] studied property testing of point configurations with respect to the *Hamming distance*. A configuration C is said to be ε-far from a property $\mathcal{P}$ in the Hamming distance if, for every $P \in \mathcal{P}$, the Hamming distance between C and P is at least $\varepsilon|C|$. In a later work, Czumaj and Sohler [4] argued that this notion of distance tells very little about the geometry behind the object. Indeed, although configuration B from Fig. 1 is obtained by applying a small perturbation to each of the red points of a convex configuration A, it is still $\frac{1}{4}$-far from Conv in the Hamming distance — in particular, B is as far from Conv as the configuration F in Fig. 1. By taking into account the Euclidian distance between points, the authors of [4] considered a distance measure between point configuration that is not sensitive to small perturbations.

In this work, we consider another distance between point configurations different from the Hamming distance. Unlike the distance defined in [4], our distance here is purely combinatorial.

Definition 1 (dist). *The distance between two point configurations C and C' of the same size is defined as*

$$\operatorname{dist}(C, C') = \min_{\iota} \frac{1}{2n^3} \sum_{(u,v,w)\in C^3} |\chi(u,v,w) - \chi(\iota(u),\iota(v),\iota(w))|, \tag{2}$$

where the minimum is taken over all bijections $\iota : C \to C'$.

Despite the fact that $\operatorname{dist}(\cdot,\cdot)$ is oblivious to the Euclidean distances between the considered points, it is interesting to note that it bears some resemblance to the geometric distance defined by Czumaj and Sohler [4] in the sense that it is also not sensitive to *small* perturbation on the point set. In our case, a perturbation is small if it flips the orientation of a small number of triples of points. For instance, for A and B as in Fig. 1, we have $\operatorname{dist}(B, \textsc{Conv}) \le \operatorname{dist}(B, A) = O(1)$, since only the triple that intersects some group of four points more than twice may have its orientation in A different from the one in B. As another example, note that configuration F from Fig. 1 is such that $\operatorname{dist}(F, A) = \Omega(1)$.

2.1 Property Testing for Configurations

An ε-*tester* $\mathcal{T}$ for a property $\mathcal{P}$ of configurations is a decision probabilistic algorithm that, when given a configuration C as input, can query the orientation $\chi(u,v,w)$ of triple of points $(u,v,w) \in C^3$. Moreover, the ε-tester $\mathcal{T}$ must satisfy the following conditions:

1. if $C \in \mathcal{P}$, then $\mathbb{P}(\mathcal{T}$ accepts $C) \ge 1 - \varepsilon$,
2. if $\operatorname{dist}(C, \mathcal{P}) > \varepsilon$, then $\mathbb{P}(\mathcal{T}$ rejects $C) \ge 1 - \varepsilon$.

If $\mathcal{T}$ accepts inputs $C \in \mathcal{P}$ with probability 1 we say $\mathcal{T}$ has one-sided error. Otherwise, we say $\mathcal{T}$ has two-sided error.

A property $\mathcal{P}$ is *testable with query complexity* $q : (0,1] \to \mathbb{N}$ if for every $\varepsilon \in (0,1]$ there is an ε-tester for $\mathcal{P}$ that queries the orientation of at most $q(\varepsilon)$ triples.

The testers considered here will perform its decision after sampling uniformly at random a set of $s = s(\varepsilon)$ vertices and querying the orientation of every triple of points in the sample. In that case we say that the tester has *sample complexity* $s(\varepsilon)$.

2.2 Testing Results

A configuration property $\mathcal{P}$ is *hereditary* if it is closed under taking subsets. The first result below states that every hereditary property $\mathcal{P}$ is testable (with two-sided error) with sample complexity polynomial on the error parameter ε^{-1}.

Moreover, this sample complexity is *universal*, that is, it does not depend on the property $\mathcal{P}$ being tested. However, the testers given by this result need to assume that the input configuration is larger than some n_0, which depends on the property $\mathcal{P}$. More precisely, n_0 depends on the so-called *blow-up parameter* $k^*_{\mathcal{P}}$ of $\mathcal{P}$ (see Sect. 3). For now, it will be enough to keep in mind that $k^*_{\mathcal{P}} : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ is a well-defined function for every hereditary property $\mathcal{P}$.

Theorem 1. *For every $\varepsilon > 0$, there is $q = \texttt{poly}(\varepsilon^{-1})$ such that every hereditary property $\mathcal{P}$ admits an ε-tester with sample complexity q for input configurations of size at least $n_0 = \texttt{poly}(\varepsilon^{-1}) k^*_{\mathcal{P}}(\texttt{poly}(\varepsilon^{-1}))$.*

As usual, we write $\texttt{poly}(t)$ for a quantity of the form $t^{O(1)}$. We are also able to prove a result stating that every hereditary property is testable with one-sided error. In that case, the sample complexity depends on the parameter $k^*_{\mathcal{P}}$.

Theorem 2. *For every $\varepsilon > 0$, every hereditary property $\mathcal{P}$ admits an ε-tester with one-sided error with sample complexity $q' = \texttt{poly}(\varepsilon^{-1}) k^*_{\mathcal{P}}(\texttt{poly}(\varepsilon^{-1}))$. Moreover, we can assume that this ε-tester is* canonical, *i.e., that when given an input C, the tester simply samples a set $S \subset C$ of size q' and accepts C if and only if $S \in \mathcal{P}$.*

3 The Blow-Up Parameter

We define the concept of *blow-ups* of configurations, which is analogous to its counterpart in [9] for permutations.

Definition 2 (k-blow-up). *Let $C \subset \mathbb{R}^2$ be a configuration and k a positive integer. A configuration C' is a k-*blow-up *of C if there is a function $\pi : C' \to C$ satisfying (i) $|\pi^{-1}(v)| = k$ for every $v \in C$ and (ii) $\chi(x, y, z) = \chi(\pi(x), \pi(y), \pi(z))$ for every x, y and $z \in C'$ with $\pi(x)$, $\pi(y)$ and $\pi(z)$ pairwise distinct. The sets $\pi^{-1}(v)$ with $v \in C$ are called the* blocks *of the blow-up C'.*

In Fig. 2 we show a 4-blow-up of the configuration N_4. We also note that both configurations A and B are 4-blow-ups of the n-point convex configuration.

For every hereditary property $\mathcal{P}$, we define the *blow-up parameter* of $\mathcal{P}$ as the function $k^*_{\mathcal{P}} : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ such that, for every $m > 0$, the integer $k^*_{\mathcal{P}}(m)$ is the least k^* satisfying the following: *if a configuration C of size $|C| = m$ admits a k^*-blow-up in $\mathcal{P}$, then C admits an ℓ-blow-up in $\mathcal{P}$ for every integer $\ell > 0$.*

Definition 3 (k^*_F, $k^*_{\mathcal{P}}$). *Let $\mathcal{P}$ be a hereditary property. For every configuration C we let*

$$k^*_{\mathcal{P}}(C) = \begin{cases} \infty, & \text{if } \forall\, k\, \exists\, k\text{-blow-up of } C \text{ in } \mathcal{P}, \\ \min\{k :\ k > 0 \text{ and } \nexists\, k\text{-blow-up of } C \text{ in } \mathcal{P}\}, & \text{otherwise.} \end{cases}$$

The blow-up parameter $k^*_{\mathcal{P}} : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ *of $\mathcal{P}$ is given by*

$$k^*_{\mathcal{P}}(m) = \begin{cases} 1, & \text{if } k^*_{\mathcal{P}}(C) = \infty \text{ for every } C \text{ of size } m, \\ \max\{k^*_{\mathcal{P}}(C) : |C| = m \text{ and } k^*_{\mathcal{P}}(C) < \infty\}, & \text{otherwise.} \end{cases} \tag{3}$$

If $\mathcal{P} = \text{Forb}(F)$ we denote $k^*_{\mathcal{P}}$ by k^*_F.

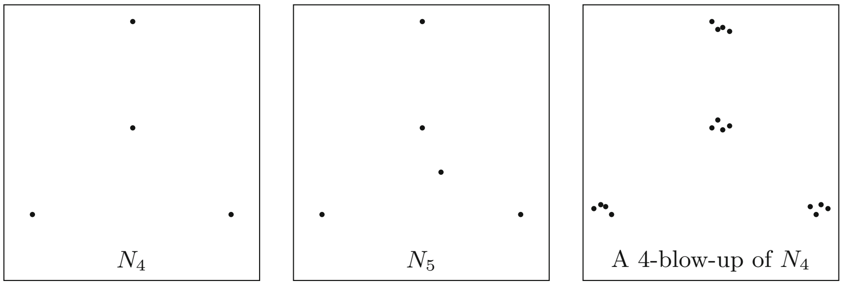


Fig. 2. The configuration N_4 and a blow-up

Example 1. Let $F = N_4$, where N_4 is as in Fig. 2. Then $\text{Forb}(N_4)$ is the set $\textsc{Conv}$ of all convex configurations. We have $k^*_F(m) = 1$ for every $m > 0$. Indeed, one can check that, for any $k > 0$ and any $C \in \textsc{Conv}$, it is always possible to construct a *convex* k-blow-up of C. Thus $k^*_F(C) = \infty$ for every $C \in \textsc{Conv}$. On the other hand, it is trivial that $k^*_F(C) = 1$ for any non-convex C.

Example 2. Let $F = N_5$, where N_5 is as in Fig. 2. Note that N_5 is isomorphic to any five point configuration in which two of them is contained in the convex hull of the other three. We claim that $k^*_F(m) \leq 2$ for every $m > 0$. Indeed, since $\textsc{Conv} \subset \text{Forb}(N_5)$, we must have $k^*_F(C) = \infty$ for every $C \in \textsc{Conv}$, by the reasoning in Example 1. On the other hand, if $C \notin \textsc{Conv}$, then $C \supset N_4$. But since every 2-blow-up of N_4 contains, in particular, two points inside the convex hull of other three points, every 2-blow-up of C must contain a copy of N_5. Thus $k^*_F(C) \leq 2$ for every $C \notin \textsc{Conv}$.

4 The Erdős–Hajnal Property and Upper Bounds for the Blow-Up Parameter k^*_F

For every configuration F, let $\text{ES}_F(r)$ be the least n such that every configuration $C \in \text{Forb}(F)$ with $|C| = n$ contains r points in convex position. Clearly, $\text{ES}_F(r) \leq \text{ES}(r)$, where $\text{ES}(r)$ is the least n such that *every* C with $|C| = n$ contains r points in convex position. By a well-known theorem of Erdős and Szekeres [6], we have that $\text{ES}(r)$ is at most exponential in r. If $\text{ES}_F(r) = \texttt{poly}(r)$, then F is said to satisfy the *Erdős–Hajnal property.* In [14], Károyi and Solymosi exhibited some configurations that satisfy the Erdős–Hajnal property. We prove the following result on the blow-up parameter k^*_F.

Theorem 3. *For every configuration F of f points, $k^*_F(m) \leq \texttt{exp}(f\ \texttt{poly}(m))$. In addition, if F satisfies the Erdős–Hajnal property, then $k^*_F(m) = \texttt{poly}(m, f)$.*

We write $\texttt{exp}(t)$ for a quantity of the form $2^{O(t)}$. Theorem 3 gives the following corollary of Theorems 1 and 2.

Corollary 1. *Assume the configuration F satisfies the Erdős–Hajnal property and let $\mathcal{P} = \mathrm{Forb}(F)$. For every $\varepsilon > 0$, property $\mathcal{P}$ admits an ε-tester with sample complexity $q = \mathtt{poly}(\varepsilon^{-1})$ for input configurations of size at least $n_0 = \mathtt{poly}(\varepsilon^{-1})$; moreover, $\mathcal{P}$ admits a one-sided canonical ε-tester with sample complexity $q = \mathtt{poly}(\varepsilon^{-1})$.*

In order to prove Theorem 3 we need the following lemma, which states that every large enough blow-up of a configuration is such that we can find 'line segments' A_i within each block of the blow-up, where a 'line segment' A_i is, roughly speaking, a collection of points in convex position for which every line defined by two of its points divides the plane in the same way.

Lemma 1. *For any integer m and configuration F with $|F| = f$, there is $k = k_1(m, F) \leq \mathrm{ES}_F(f\,\mathtt{poly}(m))$ satisfying the following. Let C be a configuration with $|C| = m$ and C' be a k-blow-up of C with blocks $B_1, \ldots, B_m$. Assume that $C' \in \mathrm{Forb}(F)$. Then there are subsets $A_i \subset B_i$ ($i \in [m]$) such that, for every $i \in [m]$,*

(a) A_i is a set $\{v_1^{(i)}, \ldots, v_f^{(i)}\}$ of f points in convex position and

(b) for every $j \in [m] \setminus \{i\}$, there is $\sigma \in \{-1, 1\}$ such that $\chi(v_a^{(i)}, v_b^{(i)}, w) = \sigma$ for every $1 \leq a < b \leq f$ and every $w \in A_j$. In particular, $\bigcup_{i=1}^m A_i$ is an f-blow-up of C.

The proof of Lemma 1 gives the bound $k_1(m, F) \leq \mathrm{ES}_F(f\,\mathtt{poly}(m)) \leq \mathtt{exp}(f\,\mathtt{poly}(m))$ for general F and, in addition, if F satisfies the Erdős–Hajnal property, then one obtains $k_1(m, F) \leq \mathtt{poly}(m, f)$. Thus to prove Theorem 3 it suffices to prove the following theorem.

Theorem 4. *For every configuration F of f points, $k_F^*(m) \leq k_1(m, F)$.*

Proof. Let $k = k_1(m, F)$. By the definition of k_F^*, we need to show that, for any configuration C with $|C| = m$, either there is no k-blow-up of C in $\mathrm{Forb}(F)$ or, for every $\ell > 0$, there is an ℓ-blow-up of C in $\mathrm{Forb}(F)$. If $C \notin \mathrm{Forb}(F)$, then clearly the former holds. We therefore consider the case in which $C \in \mathrm{Forb}(F)$.

Suppose $C \in \mathrm{Forb}(F)$ admits a k-blow-up C^k with $C^k \in \mathrm{Forb}(F)$. Fix an arbitrary $\ell > k$. We show that there is an ℓ-blow-up C^ℓ of C with $C^\ell \in \mathrm{Forb}(F)$.

We apply Lemma 1 with $C' = C^k$ and obtain sets $A_1, \ldots, A_m \subset C^k$ each of size f satisfying Lemma 1(a) and (b). Let $A_i = \{v_1^{(i)}, \ldots, v_f^{(i)}\}$ for all $i \in [m]$. One can add $\ell - f$ points to each A_i in order to obtain sets $L_1, \ldots, L_m$ such that (A) L_i is a set $\{u_1^{(i)}, \ldots, u_\ell^{(i)}\}$ of ℓ points in convex position, (B) for every $j \in [m] \setminus \{i\}$, there is $\sigma \in \{-1, 1\}$ such that $\chi(u_a^{(i)}, u_b^{(i)}, w) = \sigma$ for every $1 \leq a < b \leq \ell$ and every $w \in L_j$, and (C) the configuration $C^\ell := \bigcup_{i=1}^m L_i$ is an ℓ-blow-up of C, with blocks $L_1, \ldots, L_m$.

We claim that $C^\ell \in \mathrm{Forb}(F)$. Indeed, for a contradiction, suppose that there is a copy F' of F in C^ℓ. Let $F' = \bigcup_{i=1}^m \{w_1^{(i)}, \ldots, w_{f_i}^{(i)}\}$, where $\{w_1^{(i)}, \ldots, w_{f_i}^{(i)}\} \subset L_i$ ($0 \leq f_i \leq f$). We show that $F'' := \bigcup_{i=1}^m \{v_1^{(i)}, \ldots, v_{f_i}^{(i)}\}$ is isomorphic to

$F' = \bigcup_{i=1}^{m}\{w_1^{(i)},\dots,w_{f_i}^{(i)}\}$, whence $F \simeq F'' \subset C^k \in \mathrm{Forb}(F)$, which is a contradiction. To prove that F'' and F' are isomorphic, define the bijection ι by $\iota(v_j^{(i)}) = w_j^{(i)}$ for any $i \in [m]$ and $j \in [f_i]$. We now prove that

$$\chi(v_{j_1}^{(i_1)}, v_{j_2}^{(i_2)}, v_{j_3}^{(i_3)}) = \chi(w_{j_1}^{(i_1)}, w_{j_2}^{(i_2)}, w_{j_3}^{(i_3)}). \tag{4}$$

First, since both F' and F'' are subsets of blow-ups of C, if i_1, i_2 and i_3 are pairwise distinct, (4) holds. Secondly, since for any $i \in [m]$ both $\{w_1^{(i)},\dots,w_{f_i}^{(i)}\}$ and $\{v_1^{(i)},\dots,v_{f_i}^{(i)}\}$ are in convex position, we can make (4) hold if $i_1 = i_2 = i_3 = i$ by choosing a suitable labelling of the points. Finally, without loss of generality, assume that $i_1 = i_2$ and $i_2 \neq i_3$. In this case (4) holds because of (B) above and the fact that $A_i \subset L_i$. The claim is therefore proved. Theorem 4 follows. □

5 Proofs of Theorems 1 and 2

The main argument used to prove Theorem 1 resembles the one used by Fox and Wei [9] to show that, under a certain notion of distance, hereditary permutation properties are testable with polynomial (on the error parameter) sample complexity. It also applies the Fox–Pach–Suk Regularity Lemma for uniform semi-algebraic hypergraphs [10] to get a regular partition of configurations of points.

Definition 4. *We say that a triple $(X, Y, Z) \subset \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$ is* homogeneous *if there is $\sigma \in \{1, -1\}$ such that $\chi(x, y, z) = \sigma$ for every $(x, y, z) \in (X, Y, Z)$.*

A partition $\mathcal{V} = \{V_i\}_{i=1}^m$ is called *equitable* if $\big||V_i| - |V_j|\big| \le 1$ for any $i, j \in [m]$.

Lemma 2 ([10, Theorem 4.3]). *For every $\gamma > 0$, there exists $M = M_2(\gamma) =$ $\mathtt{poly}(1/\gamma)$ such that every configuration P, $|P| = n \ge 1/\gamma$, admits an equitable partition $\mathcal{V} = \{V_i\}_{i=1}^m$ satisfying the following*

(1) $1/\gamma \le m \le M$,
(2) At most γn^3 triples of points lie on non-homogeneous triples (V_i, V_j, V_ℓ).

*Such a partition is called a γ-*homogeneous partition *of P.* □

One consequence of Lemma 2 is that (for simplicity, say $n/m \in \mathbb{N}$) if one considers a γ-homogeneous partition $\{V_i\}_{i=1}^m$ of a configuration P of n points and an arbitrary set of points $C = \{v_i\}_{i=1}^m$ (with $v_i \in V_i$), then P is γ-close to every n/m-blow-up of C. Hence, if P is far from a hereditary property $\mathcal{P}$, it follows that every n/m-blow-up of C is not in $\mathcal{P}$. Since a large enough sample of P will contain such a set C with high probability, we get the following result.

Lemma 3. *For every $\varepsilon > 0$, there exists $M = M_2(\varepsilon/2)$, $k = k^*_{\mathcal{P}}(M)$ and $q = 2M\log(M/\varepsilon)$ satisfying the following. Let P be an order type of size $n \ge M$ and let Q be a set of q points in P chosen uniformly at random. If $\mathrm{dist}(P, \mathcal{P}) > \varepsilon$, then, with probability at least $1 - \varepsilon$, no k-blow-up of Q is in $\mathcal{P}$.*

Proof. Let $\mathcal{V} = \{V_1, \dots, V_m\}$ be an $\varepsilon/2$-homogeneous partition of P. Note that for any $i \in [m]$, $\mathbb{P}(Q \cap V_i = \emptyset) \leq (1 - 1/(2m))^q \leq (1 - 1/(2M))^q$. Then

$$\mathbb{P}(Q \cap V_i = \emptyset \text{ for some } i \in [m]) \leq M\,(1 - 1/(2M))^q \leq Me^{-q/(2M)} \leq \varepsilon.$$

Suppose the event $[Q \cap V_i = \emptyset$ for some $i \in [m]]$ does not happen and let $C = \{v_1, \dots, v_m\} \subset Q$, where $v_i \in V_i$ for $i \in [m]$. Write $t := \lceil n/m \rceil$. Assume that $\operatorname{dist}(P, \mathcal{P}) > \varepsilon$, we claim that *every* t-blow-up of C is not in $\mathcal{P}$. Indeed, let C' be an arbitrary t-blow-up of C and let $\iota : P \to C'$ be an injection that (arbitrarily) maps each point $u \in V_i$ to some point $u' \in C'$ belonging to the v_i-block — i.e., some point $u' \in \pi^{-1}(v_i)$ where $\pi : C' \to C$ is a function as in Definition 2 that witnesses C' as an t-blow-up of C. Let $C_n = \iota(P)$ be the image of ι and thus we can view $\iota : P \to C_n$ as a bijection. For every triple (u_i, u_j, u_ℓ) that belongs to a homogeneous triple (V_i, V_j, V_ℓ) we have $\chi(u_i, u_j, u_\ell) = \chi(v_i, v_j, v_\ell) = \chi(\iota(u_i), \iota(u_j), \iota(u_\ell))$. Hence, the triples that contribute to the distance between P and C_n (see Definition 2) are those that belong to non-homogeneous triples. Since $\mathcal{V}$ is an $\varepsilon/2$-homogeneous partition, it follows that

$$\operatorname{dist}(P, C_n) \leq \frac{2(\varepsilon/2)n^3}{2n^3} = \varepsilon/2.$$

Now, since $\operatorname{dist}(P, \mathcal{P}) > \varepsilon$, we get that $\operatorname{dist}(C_n, \mathcal{P}) > \varepsilon/2$, i.e., C_n is not in $\mathcal{P}$. Since $\mathcal{P}$ is hereditary, $C' \supset C_n$ is not in $\mathcal{P}$ either. Thus, it follows that *no* t-blow-up of C is in $\mathcal{P}$. If $t \geq k$, then by Definition 3, no k-blow-up of C is in $\mathcal{P}$; otherwise $t < k$, then because $\mathcal{P}$ is hereditary, no k-blow-up of C is in $\mathcal{P}$. Since $C \subset Q$ and $\mathcal{P}$ is hereditary, we conclude that no k-blow-up of Q is in $\mathcal{P}$. ◻

The previous lemma shows that if P is far from $\mathcal{P}$ then (w.h.p.) a large enough sample Q of P is such that every k-blow-up of Q is not in $\mathcal{P}$. Next, we show that if $P \in \mathcal{P}$, then (w.h.p.) a sample Q admits a k-blow-up in $\mathcal{P}$. This is a consequence of the fact that, given k and q, every large enough configuration P contains (w.h.p.) a k-blow-up of a sample Q of size q. An analogous statement is true when considering other combinatorial objects, like graphs [7], permutation [9, Lemma 3.2], etc.

Lemma 4. *For every positive integers q and k and $\varepsilon > 0$, there exists $n_0 = M_2(\varepsilon/q^3) \cdot k$ satisfying the following. If $P \in \mathcal{P}$, $|P| \geq n_0$, and Q is a set of q points in P chosen uniformly at random, then, with probability at least $1 - \varepsilon$, the configuration Q admits a k-blow-up in $\mathcal{P}$.*

Proof. Let $\mathcal{W} = \{W_i\}_{i=1}^m$ be an ε/q^3-homogeneous partition of P. The probability that Q contains a triple $(w_i, w_j, w_\ell) \in W_i \times W_j \times W_\ell$, for some non-homogeneous (W_i, W_j, W_ℓ) is at most

$$\sum_{W_i, W_j, W_\ell} q^3 \frac{|W_i|}{n} \frac{|W_j|}{n} \frac{|W_\ell|}{n} = \frac{q^3}{n^3} \sum_{W_i, W_j, W_\ell} |W_i||W_j||W_\ell| \leq \frac{q^3}{n^3} \cdot \frac{\varepsilon}{q^3} n^3 = \varepsilon.$$

Hence, with probability at least $1-\varepsilon$ every triple of points of Q lies in a homogeneous triple. Moreover $|W_i| \geq n/m \geq k$. Therefore, one can find a k-blow-up Q' of Q such that $Q' \subset P$, by arbitrarily picking k vertices from each $W_i \in \mathcal{W}$ that satisfies $|W_i \cap Q| \neq \emptyset$. Since $Q' \subset P$ and $P \in \mathcal{P}$, we must have $Q' \in \mathcal{P}$. □

Algorithm 1. Two-sided ε-tester

```
1 Let $q$ and $k$ be defined as in Lemma 3 and $n_0$ as in Lemma 4. Given an input
  configuration $P$ of size at least $n_0$, we pick a subconfiguration $Q$ of $q$ points
  uniformly at random. The $\varepsilon$-tester accepts $P$ if and only if there exists a
  $k$-blow-up of $Q$ in $\mathcal{P}$ (note that this decision can be performed in constant time,
  since the set of graphs of size $q$ that admit a $k$-blow-up in $\mathcal{P}$ can be hardcoded
  into the $\varepsilon$-tester).
```

Proof of Theorem 1. We use the tester as described in Algorithm 1, whose correctness is guaranteed by Lemmas 3 and 4. □

Once Theorem 1 is proved, Theorem 2 follows from a double sampling argument, analogous to the one used in [12, Proposition D.2] for graphs.

Algorithm 2. One-sided ε-Tester

```
1 Let $q$ and $q' = n_0$ be returned by Theorem 1 with input $\varepsilon' = \varepsilon/2$. Given an
  input configuration $P$ of size at least $n_0$, we pick a subconfiguration $Q$ of $n_0$
  points uniformly at random. The tester accepts $P$ if and only if $Q \in \mathcal{P}$.
```

Proof of Theorem 2. Let $\mathcal{P}$ be a hereditary configuration property and $\varepsilon \in (0,1]$ be fixed. We use the tester as described in Algorithm 2. Let q and $q' = n_0$ be returned by Theorem 1 with input $\varepsilon' = \varepsilon/2$ and let Acc be the set of configurations Q of size q that makes the tester from Theorem 1 accept the input when Q is sampled.

Now we show the correctness of the algorithm. If $P \in \mathcal{P}$, then it is clear that $Q \in \mathcal{P}$. We will show that if $\operatorname{dist}(P,\mathcal{P}) > \varepsilon$, then $\mathbb{P}(Q \in \mathcal{P}) < \varepsilon$. In fact, let Q' be a subconfiguration of Q of size q picked uniformly at random. Then

$$\mathbb{P}(Q' \in \textsc{Acc}) \geq \mathbb{P}(Q' \in \textsc{Acc} \mid Q \in \mathcal{P}) \cdot \mathbb{P}(Q \in \mathcal{P}) \geq (1-\varepsilon') \cdot \mathbb{P}(Q \in \mathcal{P}),$$

since otherwise the ε'-tester given by Theorem 1 would fail when given Q as input. On the other hand, since $\operatorname{dist}(P,\mathcal{P}) > \varepsilon'$ and since Q' has exactly the same distribution of a subconfiguration of size q picked uniformly at random from P, Theorem 1 (with input P) implies $\mathbb{P}(Q' \in \textsc{Acc}) < \varepsilon'$. It follows that

$$\mathbb{P}(Q \in \mathcal{P}) < \frac{\varepsilon'}{1-\varepsilon'} \leq 2\varepsilon' = \varepsilon,$$

as desired. □

6 Concluding Remark

Theorem 3 can be generalized to hereditary properties of the form Forb($\mathcal{F}$) with $\mathcal{F}$ finite. We hope to address the case in which $\mathcal{F}$ is infinite elsewhere.

Acknowledgments. The first author was supported by FAPESP (2014/18641-5). The second author was partially supported by FAPESP (2013/03447-6, 2013/07699-0), CNPq (310974/2013-5, 459335/2014-6) and NUMEC/USP (Project MaCLinC). The third author was supported by CNPq(130483/2016-8). The fourth author was supported by FAPESP (2015/15986-4), CNPq (459335/2014-6) and NUMEC/USP (Project MaCLinC).

A Proof Sketch for Lemma 1

We write by $l(x, y)$ as the line defined by two points x, y in the plane. Let l be a line and B be a set of points. Then we say l *crosses* B if there exist $b_1, b_2 \in B$ such that they are at the different sides of l, or, equivalently, the line segment defined by b_1, b_2 intersects l. Our key claim is the following.

Claim. Let $b \in B_i$, then any line $l \ni b$ crosses at most *one* other block B_j.

Proof. Assume that $l \ni b$ crosses both B_j and $B_{j'}$. Pick the points $u, u' \in B_j$, $v, v' \in B_{j'}$ such that u and v are at the same side of l, and u', v' are at the other side of l. Consider two lines $l(u, b)$ and $l(u', b)$. These two lines partition the plane into four regions. By the definition of the blow-up, we know that $\chi(b, u, v) = \chi(b, u, v')$ and $\chi(b, u', v) = \chi(b, u', v')$. Thus, v and v' must be in the same region. Moreover, since v and v' must be at the different sides of l, they must be both in one of the two regions that intersect l. Then note that $\chi(b, u, v) \neq \chi(b, u', v')$ holds, which is a contradiction.

Now we start the sketch of the proof. For any $i \in [m]$, we find a largest subset $C_i \subset B_i$ in convex position. By the definition of ES_F, each C_i has size at least $f\texttt{poly}(m)$. Let $C_i = \{v_1^{(i)}, \ldots, v_p^{(i)}\}$ where the points are ordered along the convex polygon under clockwise order. We now color each line segment $v_j^{(i)} v_{j+1}^{(i)}$ as follows:

- if $l(v_j^{(i)}, v_{j+1}^{(i)})$ crosses C_p for some $c \in [m]$, $c \neq i$, then give $v_j^{(i)} v_{j+1}^{(i)}$ color p;
- otherwise, we rotate the line $l(v_j^{(i)}, v_{j+1}^{(i)})$ around $v_{j+1}^{(i)}$ clock-wise. Give $v_j^{(i)} v_{j+1}^{(i)}$ color p if C_p is the first block that the line crosses.

Note that the coloring is well-defined because of the claim. In total this defines a coloring with m colors, and note that each color class consists of two (possibly empty) sets of consecutive line segments of the convex polygon C_i. Thus by the Pigeonhole principle, there is a subset $C'_i = \{u_1^{(i)}, \ldots, u_q^{(i)}\} \subset C_i$ such that $|C'_i| \geq f\texttt{poly}(m)$, and all $(u_r^{(i)}, u_{r+1}^{(i)})$ for $1 \leq r < q$ have the same color. We repeat this process for all $i \in [m]$.

Next we consider C'_1 and all lines $(u^{(i)}_{q/2}, u^{(i)}_{q/2+1})$ for every index i such that C'_i received color 1. These lines divide C'_1 in at most $\texttt{poly}(m)$ regions. Hence, by averaging, we can find a subset $D'_1 \subset C'_1$ of $2f$ points such that for every such i, either $\{u^{(i)}_1, \ldots, u^{(i)}_{q/2}\}$ or $\{u^{(i)}_{q/2+1}, \ldots, u^{(i)}_q\}$ are entirely on the same side of D'_1. Thus, abusing the notation, we pick either $\{u^{(i)}_1, \ldots, u^{(i)}_{q/2}\}$ or $\{u^{(i)}_{q/2+1}, \ldots, u^{(i)}_q\}$ as our new C'_i, and we still have $|C'_i| \geq f\ \texttt{poly}(m)$.

We can continue the operation for every C'_r $(1 \leq r \leq m)$ (but considering only the points survived from previous iterations). This gives sets $A_1, \ldots, A_m$ each of size f satisfying Lemma 1(a) and (b), because during the whole process, a set C'_r is shrunk at most twice: by a factor of $\texttt{poly}(m)$ when it plays the role of C'_j, or by a factor of 2 when it plays the role of C'_i. □

References

1. Bland, R.G., Las Vergnas, M.: Orientability of matroids. J. Comb. Theory Ser. B **24**(1), 94–123 (1978)
2. Bokowski, J., Sturmfels, B.: Computational Synthetic Geometry. Lecture Notes in Mathematics. Springer, Berlin (1989). https://doi.org/10.1007/BFb0089253
3. Chen, X., Freilich, A., Servedio, R.A., Sun, T.: Sample-based high-dimensional convexity testing (June 2017). arXiv:1706.09362
4. Czumaj, A., Sohler, C.: Property testing with geometric queries (extended abstract). In: auf der Heide, F.M. (ed.) ESA 2001. LNCS, vol. 2161, pp. 266–277. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_22
5. Czumaj, A., Sohler, C., Ziegler, M.: Property testing in computational geometry (extended abstract). In: Paterson, M.S. (ed.) ESA 2000. LNCS, vol. 1879, pp. 155–166. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45253-2_15
6. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compos. Math. **2**, 463–470 (1935)
7. Erdős, P., Simonovits, M.: Supersaturated graphs and hypergraphs. Combinatorica **3**(2), 181–192 (1983). https://doi.org/10.1007/BF02579292
8. Ergün, F., Kannan, S., Kumar, S.R., Rubinfeld, R., Viswanathan, M.: Spot-checkers. J. Comput. Syst. Sci. **60**(3), 717–751 (2000). https://doi.org/10.1006/jcss.1999.1692. 30th Annual ACM Symposium on Theory of Computing (Dallas, TX 1998)
9. Fox, J., Wei, F.: Fast property testing and metrics for permutations (2016). arXiv:161101270
10. Fox, J., Pach, J., Suk, A.: A polynomial regularity lemma for semialgebraic hypergraphs and its applications in geometry and property testing. SIAM J. Comput. **45**(6), 2199–2223 (2016). https://doi.org/10.1137/15M1007355
11. Goldreich, O. (ed.): Property Testing: Current Research and Surveys. LNCS, vol. 6390. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16367-8
12. Goldreich, O., Trevisan, L.: Three theorems regarding testing graph properties. Random Struct. Algorithms **23**(1), 23–57 (2003). https://doi.org/10.1002/rsa.10078
13. Goodman, J.E., Pollack, R.: Multidimensional sorting. SIAM J. Comput. **12**(3), 484–507 (1983). https://doi.org/10.1137/0212032

14. Károlyi, G., Solymosi, J.: Erdős-Szekeres theorem with forbidden order types. J. Combin. Theory Ser. A **113**(3), 455–465 (2006). https://doi.org/10.1016/j.jcta.2005.04.006
15. Richter-Gebert, J., Ziegler, G.M.: Oriented matroids. In: Handbook of Discrete and Computational Geometry, CRC Press Series Discrete Mathematics Application, pp. 111–132. CRC, Boca Raton (1997)
16. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)

Maximal and Convex Layers of Random Point Sets

Meng He[(✉)], Cuong P. Nguyen, and Norbert Zeh

Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
{mhe,nzeh}@cs.dal.ca, cn536386@dal.ca

Abstract. We study two problems concerning the maximal and convex layers of a point set in d dimensions. The first is the average-case complexity of computing the first k layers of a point set drawn from a uniform or component-independent (CI) distribution. We show that, for $d \in \{2, 3\}$, the first $n^{1/d-\epsilon}$ maximal layers can be computed using $dn + o(n)$ scalar comparisons with high probability. For $d \geq 4$, the first $n^{1/2d-\epsilon}$ maximal layers can be computed within this bound with high probability. The first $n^{1/d-\epsilon}$ convex layers in 2D, the first $n^{1/2d-\epsilon}$ convex layers in 3D, and the first $n^{1/(d^2+2)}$ convex layers in $d \geq 4$ dimensions can be computed using $2dn + o(n)$ scalar comparisons with high probability. Since the expected number of maximal layers in 2D is $2\sqrt{n}$, our result for 2D maximal layers shows that it takes $dn + o(n)$ scalar comparisons to compute a $1/n^{\epsilon}$-fraction of all layers in the average case. The second problem is bounding the expected size of the kth maximal and convex layer. We show that the kth maximal and convex layer of a point set drawn from a continuous CI distribution in d dimensions has expected size $O(k^d \log^{d-1}(n/k^d))$.

Keywords: Maximal layers · Skyline · Convex layers
Average-case analysis

1 Introduction

Maximal and convex layers are fundamental geometric structures with applications for example in data mining [5], pattern recognition and statistics [8,15]. A point p *dominates* another point q if p is no less than q in any dimension and p is greater than q in at least one dimension. The *skyline* (*first maximal layer*) of a d-dimensional point set S is the set of all points in S not dominated by any other point in S. A point $p \in S$ belongs to the *convex hull* (*first convex layer*) of S if there exists a $(d-1)$-dimensional hyperplane through p that has all points of S on the same side. For $k > 1$, the *kth maximal or convex layer* is the skyline or convex hull of the subset of S obtained by removing the first $k-1$ maximal or convex layers, respectively.

Computing maximal and convex layers are problems that have been studied extensively. A classical result of Kung et al. [12] shows that the skyline of a point set in 2D or 3D can be found in $O(n \log n)$ time; for any constant $d \geq 4$, the cost in d dimensions is $O(n \log^{d-2} n)$. The convex hull of a 2D or 3D point set can also be found in $O(n \log n)$ time [3], while the cost of finding the convex hull in $d \geq 4$ dimensions is $\Theta(n^{\lfloor d/2 \rfloor})$ in the worst case [7]. A simple adversary argument shows that, in the worst case, $\Omega(n \log n)$ comparisons between scalars are necessary to compute the skyline or convex hull in d dimensions for any $d \geq 2$. For component-independent (CI) point distributions, on the other hand, expected linear-time algorithms exist, where a point distribution is component-independent if it is continuous and the coordinates of each point are chosen independently. The algorithm of Bentley et al. [1] uses $dn + o(n)$ scalar comparisons in expectation to find the skyline of a point set in d dimensions. For the convex hull, they presented an algorithm that uses $2dn + o(n)$ expected scalar comparisons for $d \in \{2, 3\}$. For $d \geq 4$, they presented an algorithm that finds a superset of the convex hull of expected size $O(\log^{d-1} n)$ using $2dn + o(n)$ expected scalar comparisons. They also proved that dn scalar comparisons is a lower bound for computing either the skyline or convex hull.

All maximal layers of a point set can be computed in $O(n \log n)$ time in 2D [4] and 3D [6]. For $d \geq 4$, no optimal algorithm for computing multiple maximal layers is known. The convex layers of a point set in 2D can be computed in $O(n \log n)$ time [8]. For $d \geq 3$, no optimal algorithm for computing multiple convex layers is known. Nielsen [15] presented an output-sensitive algorithm for finding the first k convex or maximal layers of a 2D point set in $O(n \log h_k)$ time, where h_k is the number of points in these layers.

One of the key ingredients of Bentley et al.'s skyline and convex hull algorithms [1] is the ability to quickly identify a small subset of points that is likely to contain all skyline or convex hull points. The skyline or convex hull can then be computed by applying one of the algorithms above to this subset of points. Such a small subset can exist only if the skyline or convex hull is small. Bentley et al. [2] proved that the expected size of the skyline or the expected number of vertices of the convex hull over an arbitrary CI distribution is $O(\log^{d-1} n)$. Note that the work on the expected complexity of geometric structures, including that of Bentley et al. [2], is of independent interest. Many other problems have also been defined and studied under similar assumptions. For instance, Dalal [9] shows that the expected number of convex layers is $\Theta(n^{2/(d+1)})$ for a set of n points independently chosen from a uniform distribution inside any bounded, nonempty region in $\mathbb{R}^d$. We refer to Okabe et al. [16] for a review of many problems in this area.

Our results. We extend Bentley et al.'s results [1] to multiple layers and strengthen the algorithm analysis by proving high-probability bounds on the number of scalar comparisons. Our first main result is a reduction that allows us to obtain an algorithm that computes the first k maximal or convex layers using $dn + o(n)$ or $2dn + o(n)$ expected scalar comparisons, respectively, given an algorithm that computes these layers using $O(k^c n^{1+\epsilon})$ scalar comparisons in the worst case. The exact bound on k is given in the following theorem.

Theorem 1. *Let S be a set of n points drawn from an arbitrary CI distribution in d dimensions. Suppose there is an algorithm M that can compute the first k maximal (or convex) layers of S using $O(k^c n^{1+\epsilon})$ scalar comparisons in the worst case, where c and ϵ are constants with $c \geq 0$ and $0 < \epsilon < \frac{1}{(c+1)d}$. Then the first $\kappa = n^{\frac{1}{(c+1)d} - \epsilon}$ maximal (or convex) layers of S can be computed using $dn + o(n)$ (or $2dn + o(n)$) expected scalar comparisons, and the actual number of comparisons is within the same bounds with probability $1 - o(n^{-n^{\epsilon'}})$ for any $\epsilon' \in (0, (c\epsilon + \frac{\epsilon^2}{2(\epsilon+1)})d)$.*

To achieve this result, our main strategy is to generalize the algorithms of Bentley et al. [1] to compute more than one maximal or convex layer. While it is not difficult to generalize the algorithms themselves, it is more challenging to analyze their running times. To perform the analysis, our key strategy is to further conceptually subdivide some objects defined by these algorithms into even smaller objects, such that a layer must contain a point inside a particular smaller object with high probability. These constructs may be of general interest, as they may be useful to the tasks of performing some other similar analysis over multiple layers of the given point set.

The existing algorithms discussed previously allow us to find the first k maximal layers using $O(n^{1+\epsilon})$ comparisons for $d \in \{2, 3\}$ and using $O(kn^{1+\epsilon})$ comparisons for $d \geq 4$. The first k convex layers can be computed using $O(n^{1+\epsilon})$ comparisons in 2D and using $O(kn^{1+\epsilon})$ comparisons in 3D. Thus, we obtain the following corollary of Theorem 1:

Corollary 1. *Let S be a set of n points drawn from an arbitrary CI distribution in d dimensions. If $d \in \{2, 3\}$, the first $n^{\frac{1}{d} - \epsilon}$ maximal layers of S can be computed using $dn + o(n)$ expected scalar comparisons. If $d \geq 4$, the first $n^{\frac{1}{2d} - \epsilon}$ maximal layers can be computed using this expected number of scalar comparisons. If $d = 2$, the first $n^{\frac{1}{d} - \epsilon}$ convex layers of S can be computed using $2dn + o(n)$ expected scalar comparisons. If $d = 3$, the first $n^{\frac{1}{2d} - \epsilon}$ convex layers can be computed using this number of expected scalar comparisons. In all these cases, the actual number of comparisons is within the same upper bounds on the expected number of comparisons with probability $1 - o(n^{-n^{\epsilon'}})$.*

Our results are the first that show that more than one maximal or convex layer can be computed using the optimal number of scalar comparisons on random point sets up to lower order terms and, in the case of convex hull, up to a constant factor of 2. With the exception of a high-probability analysis of an alternative skyline algorithm by Bentley et al. [1] provided by Golin [11], only expected bounds on the number of scalar comparisons were known even for computing only the first convex or maximal layer.

The number of maximal layers of a point set S is the length of a longest monotonically increasing subsequence (LMIS) of the sequence of y-coordinates of the points in S sorted by their x-coordinates. If S is drawn from a CI distribution, this sequence of y-coordinates is a uniform random permutation of the y-coordinates. Thus, by a result of [10], the expected length of an LMIS of this sequence, and thus the number of maximal layers of S approaches $2\sqrt{n}$ as n approaches infinity. Therefore, for $d = 2$, our algorithm finds a $1/n^{\epsilon}$-fraction of all maximal layers in the average case using the optimal number of scalar comparisons up to lower-order terms.

For $d \geq 4$ dimensions, no convex hull algorithm using expected $2dn + o(n)$ comparisons on random point sets was known, and we cannot satisfy the condition of Theorem 1 even for $k = 1$ since computing the convex hull takes $\Theta(n^{\lfloor d/2 \rfloor})$ time in the worst case. However, the construction that proves Theorem 1 can be combined with the $\Theta(n^{\lfloor d/2 \rfloor})$-time convex hull algorithm to obtain the following theorem:

Theorem 2. *Let S be a set of n points in $d \geq 4$ dimensions drawn from an arbitrary CI distribution. For any $k \leq n^{1/(d^2+2)}$, the first k convex layers of S can be found using $2dn + o(n)$ scalar comparisons with probability $1 - O(\frac{1}{n^{1/d-\epsilon}})$, for any $\epsilon > 0$.*

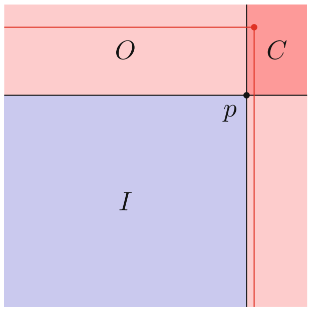
This result is the first that computes multiple convex layers in four or higher dimension in linear time with high probability.

Our second main result bounds the size of the kth maximal or convex layer of a d-dimensional point set. Previously, only bounds on the expected size of the *first* maximal or convex layer were known.

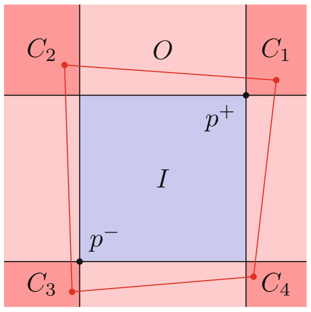
Theorem 3. *For any point set S drawn from a continuous CI distribution in d dimensions, the kth maximal or convex layer has expected size $O(k^d \log^{d-1}(n/k^d))$.*

2 Algorithm Overview

Bentley et al.'s algorithms [1] for computing the skyline or convex hull of a point set S using expected $dn+o(n)$ or $2dn+o(n)$ comparisons uses the following simple idea: Find a rectangular *inner region* I that is expected to contain almost all points in S and is likely to be completely below the skyline of S or inside the convex hull of S. See Fig. 1. In particular, with high probability, the points in S that belong to the skyline or convex hull are all contained in the *outer region* $O = \mathbb{R}^d \setminus I$. The algorithm partitions the point set S into two subsets $S_I = S \cap I$ and $S_O = S \cap O$ and computes the skyline or convex hull L_O of S_O using some standard skyline or convex hull algorithm, which takes $o(n)$ time in expectation because S_O is small. Finally, the algorithm checks whether certain subregions of O (C and $C_1, \ldots, C_4$, respectively, in Fig. 1) each contain at least one point of S_O. If so, I is completely below or inside L_O, which implies that L_O is also the skyline or convex hull of S because no point in $S \setminus S_O = S_I \subseteq I$ can be on the maximal layer or convex hull of S. Thus, the algorithm terminates in this case. Otherwise, the algorithm runs a standard skyline or convex hull algorithm on S to compute the skyline or convex hull of S. While this is costly, this happens



(a) Maximal layers



(b) Convex layers

Fig. 1. The inner and outer regions used in Bentley et al.'s [1] and our algorithm illustrated for the 2D case. I is shaded blue. O is shaded pink, including the darker regions, which are the corners that are tested by the algorithm whether they contain a point not on the first k maximal or convex layers. As illustrated in red, any point in C dominates I in the case of maximal layers; in the case of convex layers, the convex hull of any four points in $C_1, \ldots, C_4$ encloses I. (Color figure online)

infrequently because I is likely to be below the skyline or inside the convex hull of S, so the expected cost of this final step is again $o(n)$.

For the skyline algorithm, $I = (-\infty, x_1(p)] \times (-\infty, x_2(p)] \times \cdots \times (-\infty, x_d(p)]$, where p is an appropriate point and $x_i(p)$ denotes the ith coordinate of p, so the partition into S_I and S_O can be obtained using dn scalar comparisons. For the convex hull algorithm, $I = [x_1(p^-), x_1(p^+)] \times [x_2(p^-), x_2(p^+)] \times \cdots \times [x_d(p^-), x_d(p^+)]$ for an appropriate pair of corner points (p^-, p^+), so the partition can be performed using $2dn$ scalar comparisons. The corner points of I can be found without comparisons by setting $x_i(p^-) = \epsilon$ and $x_i(p) = x_i(p^+) = 1 - \epsilon$ for all $1 \le i \le d$ and some appropriate value $\epsilon > 0$. At least this is the case for points distributed uniformly at random in the unit hypercube. For an arbitrary CI distribution, p, p^-, and p^+ can each be found using $dn + o(n)$ scalar comparisons using randomized linear-time selection. The partitioning of S into S_I and S_O can be done as part of the selection process without incurring any additional comparisons. We discuss this in more detail as part of our high-probability analysis and extension to multiple layers (Lemmas 1 and 4). Overall, the expected cost of the algorithm is $dn + o(n)$ or $2dn + o(n)$ comparisons for finding p or p^- and p^+ and computing the partition of S into S_I and S_O plus $o(n)$ expected comparisons for computing the maximal layer or convex hull.

To extend Bentley et al.'s result [1] to multiple maximal or convex layers, we need to show that there exists a point p or a pair of points (p^-, p^+) that defines inner and outer regions I and O as above such that, again, almost all points in S are inside I and the first k layers are unlikely to intersect I. To achieve a running time of $dn + o(n)$ or $2dn + o(n)$ *with high probability*, we also need to strengthen the analysis of Bentley et al. [1] to (a) show that these points can be found using $dn + o(n)$ or $2dn + o(n)$ scalar comparisons with high probability and (b) with high probability, I does not intersect the first k layers.

Since the proofs are slightly simpler, we present our result for maximal layers first. Then, in Sect. 4, we argue that the same approach, with minor modifications, can also be used to compute convex layers.

3 Maximal Layers

Throughout this section, we use $p \nearrow q$ to indicate that q dominates p. Given a point set S drawn from a CI distribution $\mathcal{D}$ and some value $\tau \in [0, 1]$, we call a point p a *τ-pivot* of S if, for all $1 \leq i \leq d$ and any point p' chosen uniformly at random from S, $P[x_i(p') \geq x_i(p)] = \tau$; recall that $x_i(p)$ denotes the ith coordinate of point p. Point p is not necessarily in S. We first prove the following lemma on locating p.

Lemma 1. *Let S be a point set drawn from a CI distribution. For any value $t > 0$, any value $\tau \in (0, n^{-t}] \cup [1 - n^{-t}, 1)$, and any constant $\epsilon' \in (0, 1)$, a τ-pivot p and a partition of S into two subsets $S_I = S \cap I$ and $S_O = S \cap (\mathbb{R}^d \setminus I)$ can be computed using $dn + o(n)$ scalar comparisons in expectation and with probability at least $1 - o\left(n^{-n^{\epsilon'}}\right)$, where I is the region dominated by p.*

Proof. If S is drawn uniformly at random from the unit hypercube, then $p = (1 - \tau, \ldots, 1 - \tau)$ is a τ-pivot and can be found without any comparisons. The partition of S into S_I and S_O can be computed by deciding for each point whether it is dominated by p (and thus belongs to S_I) or not (and thus belongs to S_O). This takes d comparisons per point in S, incurring dn comparisons in total, that is, the lemma holds in the worst case for a uniform random distribution. For an arbitrary CI distribution, set $x_i(p)$ to be the (τn)th largest coordinate in dimension i among the points in S. Then p is a τ-pivot. Each value $x_i(p)$ can be found using $n + o(n)$ scalar comparisons in expectation and with probability at least $1 - o\left(n^{-n^{\epsilon'}}\right)$ using a simplified version of LazySelect [14]; we omit the details due to page constraints. In the process, every point in S is tagged as having ith coordinate less than or equal to, or greater than $x_i(p)$. Doing this for all d dimensions produces p and takes $dn + o(n)$ scalar comparisons in expectation and with probability at least $1 - o\left(n^{-n^{\epsilon'}}\right)$. The partition of S into S_I and S_O is then obtained without additional scalar comparisons by collecting all points tagged as greater than p in at least one dimension into S_O, and the remaining points into S_I. □

The following observation and lemmas are needed for our proof of Theorem 1.

Observation 4. *Let p be a τ-pivot of S and consider the corresponding partition of S into subsets S_I and S_O as in Lemma 1. If there exist $k + 1$ points $p_1, p_2, \ldots, p_{k+1}$ in S_O such that $p \nearrow p_{k+1} \nearrow \cdots \nearrow p_1$, then the first k maximal layers of S and S_O are identical and p_{k+1} is not part of these layers.*

Lemma 2. *Let S be a point set drawn from a CI distribution, let $0 < \epsilon_1 < \epsilon_2 < 1$ be constants, let $\frac{\tau}{k+1} \geq n^{(\epsilon_2 - 1)/d}$, and let $h_0, h_1, \dots, h_{k+1}$ be $k+2$ points such that h_j is a $\left(\frac{j}{k+1}\tau\right)$-pivot of S for all $0 \leq j \leq k+1$. Then with probability at least $1 - o\left(n^{-n^{\epsilon_1}}\right)$, each hyperrectangle H_j defined by points h_{j-1} and h_j, for $1 \leq j \leq k+1$, contains a point $p_j \in S$. These points satisfy $p_{k+1} \nearrow p_k \nearrow \cdots \nearrow p_1$.*

Proof. Consider an arbitrary hyperrectangle H_j. Since h_{j-1} is a $\left(\frac{j-1}{k+1}\tau\right)$-pivot and h_j is a $\left(\frac{j}{k+1}\tau\right)$-pivot, each point $p \in S$ satisfies $x_i(h_{j-1}) \leq x_i(p) \leq x_i(h_j)$ with probability $\frac{\tau}{k+1}$ for each $1 \leq i \leq d$. Since the coordinates are chosen independently, $p \in H_j$ with probability $\left(\frac{\tau}{k+1}\right)^d$. Thus, $E(|H_j \cap S|) = \left(\frac{\tau}{k+1}\right)^d n$. Since $|H_j \cap S|$ is the sum of independent Bernoulli random variables, the Chernoff bound states that $P(H_j \cap S = \emptyset) < e^{-(\tau/(k+1))^d n/4}$ and the probability that there exists an index $1 \leq j \leq k+1$ such that $H_j \cap S = \emptyset$ is less than $(k+1)e^{-(\tau/(k+1))^d n/4}$. For $\frac{\tau}{k+1} \geq n^{(\epsilon_2-1)/d}$, this is bounded by $(k+1)e^{-n^{\epsilon_2}/4} \leq n^{1-n^{\epsilon_2}/(4 \ln n)} = o\left(n^{-n^{\epsilon_1}}\right)$ for any $\epsilon_1 < \epsilon_2$ because $k+1 \leq n$. ☐

Lemma 3. *Let S be a point set drawn from a CI distribution, let $0 < \epsilon_1 < \epsilon_2 < 1$ be constants, let $\tau \geq n^{\epsilon_2 - 1}$, let p be a τ-pivot of S, let $S_I \subseteq S$ be the set of points dominated by p, and let $S_O = S \setminus S_I$. Then $E(|S_O|) \leq d\tau n$ and $P(|S_O| > 2d\tau n) = o\left(n^{-n^{\epsilon_1}}\right)$.*

Proof. We can cover the outer region O with d halfspaces $B_1, B_2, \dots, B_d$, where $B_i = \{p' \in \mathbb{R}^d \mid x_i(p') \geq x_i(p)\}$. Since a point $p' \in S$ satisfies $x_i(p') \geq x_i(p)$ with probability τ, we have $E(|B_i \cap S|) = \tau n$ and $E(|S_O|) \leq \sum_{i=1}^d E(|B_i \cap S|) = d\tau n$. Since $|B_i \cap S|$ is the sum of independent Bernoulli random variables, the Chernoff bound states that $P(|B_i \cap S| > 2\tau n) < e^{-\tau n/3} \leq n^{-n^{\epsilon_2}/(3 \ln n)} = o\left(n^{-n^{\epsilon_1}}\right)$. Thus, $P(|S_O| > 2d\tau n) \leq \sum_{i=1}^d P(|B_i| > 2\tau n) = o\left(dn^{-n^{\epsilon_1}}\right) = o\left(n^{-n^{\epsilon_1}}\right)$. ☐

Proof (Proof of Theorem 1 *(*Maximal Layers*)).* Our algorithm finds a τ-pivot p of S, partitions S into S_I and S_O, computes the first k maximal layers of S_O using M, and checks whether there exists a point in S_O that is not on the computed maximal layers but dominates p. If this test succeeds, then the maximal layers of S and S_O are the same, so the algorithm reports the computed maximal layers. Otherwise, it runs M on S to compute the first k maximal layers of S.

We prove that this algorithm uses $dn + o(n)$ scalar comparisons with high probability. The analysis of the expected number of comparisons is analogous. The number of comparisons the algorithm performs is $dn + o(n)$ if (a) computing p and partitioning S into S_I and S_O takes $dn + o(n)$ comparisons, (b) running algorithm M on S_O incurs $o(n)$ comparisons, and (c) there exists a point in S_O that is not on the first k maximal layers and dominates p, that is, the fallback option of running M on the entire point set S is not invoked. Thus, it suffices to bound the probability that any of these three conditions fails.

By Lemma 1, (a) fails with probability $o\left(n^{-n^{\epsilon'}}\right)$, for any $\epsilon' \in (0,1)$, as long as $\tau = n^{-t}$ for some $t > 0$. Running algorithm M on S_O incurs $o(n)$ scalar comparisons if $|S_O| = o\left(n^{1/(1+\epsilon)}\right)/k^c$. By Lemma 3, $|S_O| \leq 2d\tau n$ with probability

$1-o\Big(n^{-n^{\epsilon'}}\Big)$ as long as $\tau \geq n^{\epsilon_2 - 1}$ for some $\epsilon_2 > \epsilon'$. Therefore, (b) fails with probability $o\Big(n^{-n^{\epsilon'}}\Big)$ as long as $\tau n = o\big(n^{1/(1+\epsilon)}\big)/k^c$ and $\tau \geq n^{\epsilon_2-1}$. By Observation 4 and Lemma 2, (c) fails with probability $o\Big(n^{-n^{\epsilon'}}\Big)$ as long as $\frac{\tau}{k+1} \geq n^{(\epsilon_2-1)/d}$ for some $\epsilon_2 > \epsilon'$. Thus, the probability that any of these three conditions fails is $o(n^{-n^{\epsilon'}})$, provided we can choose τ so that the above constraints are satisfied.

First observe that $\epsilon_2 - 1 < 0$. Thus, $\tau \geq n^{\epsilon_2-1}$ if $\frac{\tau}{k+1} \geq n^{(\epsilon_2-1)/d}$, so we have to choose a value of $\tau = n^{-t}$, for some $t > 0$, such that $\frac{\tau}{k+1} \geq n^{(\epsilon_2-1)/d}$ and $\tau n = o\big(n^{1/(1+\epsilon)}\big)/k^c$. The last two constraints imply that $k^{c+1} = o(n^{-\epsilon/(\epsilon+1)+(1-\epsilon_2)/d})$ or $k = o(n^{\frac{1}{(c+1)d}-\epsilon+\delta})$ where $\delta = \epsilon - \frac{\epsilon_2}{(c+1)d} - \frac{\epsilon}{(\epsilon+1)(c+1)}$. For any $\epsilon_2 < \big(c\epsilon + \frac{\epsilon^2}{2(\epsilon+1)}\big)d$, we have $\delta > 0$, that is, we can compute up to $n^{\frac{1}{(c+1)d}-\epsilon}$ maximal layers and, since $(c\epsilon + \frac{\epsilon^2}{2(\epsilon+1)}) > 0$, we can choose values ϵ' and ϵ_2 such that $0 < \epsilon' < \epsilon_2 < (c\epsilon + \frac{\epsilon^2}{2(\epsilon+1)})d$. It remains to choose τ. We have $\tau n = o\big(n^{1/(1+\epsilon)}\big)/k^c$ if $t > \frac{\epsilon}{1+\epsilon} + \frac{c}{(c+1)d} - \epsilon c$. To satisfy $\frac{\tau}{k+1} \geq n^{(\epsilon_2-1)/d}$, we need $t = -\log_n \tau \leq -\log_n(k+1) - \frac{\epsilon_2 - 1}{d}$. To compute the first $n^{\frac{1}{(c+1)d}-\epsilon}$ maximal layers, we replace k by $n^{\frac{1}{(c+1)d}-\epsilon}$ in this inequality, and it holds for large enough n if t is a constant and $t < \epsilon - \frac{1}{(c+1)d} - \frac{\epsilon_2-1}{d}$, which is true as long as t is a constant satisfying $t < \epsilon - \frac{1}{(c+1)d} - \epsilon c - \frac{\epsilon^2}{2(1+\epsilon)} + \frac{1}{d}$ because $\epsilon_2 < (c\epsilon + \frac{\epsilon^2}{2(\epsilon+1)})d$. It is easy to verify that $\frac{\epsilon}{1+\epsilon} + \frac{c}{(c+1)d} - \epsilon c < \epsilon - \frac{1}{(c+1)d} - \epsilon c - \frac{\epsilon^2}{2(1+\epsilon)} + \frac{1}{d}$. Thus, we can choose a value of t that satisfies both constraints and set $\tau = n^{-t}$. In addition, since $\epsilon < \frac{1}{(c+1)d}$, we have $\frac{\epsilon}{1+\epsilon} + \frac{c}{(c+1)d} - \epsilon c > 0$, that is, $t > 0$. □

4 Convex Layers

Convex Layers in Two and Three Dimensions: To apply the framework from Sect. 3 to compute convex layers, we need to extend the notion of dominance to the 2^d possible quadrants of a point in $\mathbb{R}^d$. We identify each quadrant using a *sign vector* $\sigma \in \{+1, -1\}^d$. We say a point $q \in \mathbb{R}^d$ σ-*dominates* another point $p \in \mathbb{R}^d$, written as $p \nearrow_\sigma q$ if $\sigma \circ q$ dominates $\sigma \circ p$, where $p \circ q$ is the Hadamard product: $p \circ q = (x_1(p), x_2(p), \ldots, x_d(p)) \circ (x_1(q), x_2(q), \ldots, x_d(q)) = (x_1(p)x_1(q), x_2(p)x_2(q), \ldots, x_d(p)x_d(q))$. We call a point p a (τ, σ)-*pivot* of S if, for all $1 \leq i \leq d$ and any point p' chosen uniformly at random from S, $P(x_i(\sigma)x_i(p') \geq x_i(\sigma)x_p(p)) = \tau$. Note that $\mathbf{1}$-dominance is the same as normal dominance, a $(\tau, \mathbf{1})$-pivot is just a τ-pivot, and a $(\tau, -\mathbf{1})$-pivot is a $(1-\tau)$-pivot, where $\mathbf{1} = (1, \ldots, 1)$ and $-\mathbf{1} = (-1, \ldots, -1)$. A pair of points (p^-, p^+), where $0 < \tau < 1/2$, p^{-1} is a $(\tau, -\mathbf{1})$-pivot, and p^+ is a $(\tau, \mathbf{1})$-pivot, divides $\mathbb{R}^d$ into an *inner region* I containing all points in $\mathbb{R}^d$ that dominate p^- and are dominated by p^+, and an *outer region* $O = \mathbb{R}^d \setminus I$; see Fig. 1. Similar to maximal layers, we define $S_I = S \cap I$ and $S_O = S \cap O$. The corners of I are the points $\{p^\sigma \mid \sigma \in \{+1, -1\}^d\}$, where $p^\sigma = \frac{1}{2}((\mathbf{1} + \sigma) \circ p^+ + (\mathbf{1} - \sigma) \circ p^-)$. Since S is drawn from a CI distribution, each such corner p^σ is a (τ, σ)-pivot of S.

Our algorithm finds (p^-, p^+), partitions S into S_I and S_O, computes the first k convex layers of S_O using M, and checks whether, for every $\sigma \in \{+1, -1\}^d$, there exists a point in S_O that is not on the computed convex layers but σ-dominates p^σ. If this test succeeds, then the convex layers of S and S_O are the same, so the algorithm reports the computed convex layers. Otherwise, it runs M on S to compute the first k convex layers of S. To analyze this algorithm, we first prove the following lemmas and observation.

Lemma 4. *Let S be a point set drawn from a CI distribution. For any value $t > 0$, any value $\tau \in (0, n^{-t}]$, and any constant $\epsilon' \in (0, 1)$, a pair of points (p^-, p^+) such that p^- is a $(\tau, -\mathbf{1})$-pivot of S and p^+ is a $(\tau, \mathbf{1})$-pivot of S and a partition of S into two subsets $S_I = S \cap I$ and $S_O = S \cap O$ can be computed using $2dn + o(n)$ scalar comparisons in expectation and with probability at least $1 - o\left(n^{-n^{\epsilon'}}\right)$.*

Proof. Since p^+ is a τ-pivot and p^- is a $(1 - \tau)$-pivot of S, we can find these two points using the claimed number of scalar comparisons by applying Lemma 1 twice. In the case of an arbitrary CI distribution, the selection of the coordinates of p^- and p^+ also tags each point as having ith coordinate less than $x_i(p^-)$, between $x_i(p^-)$ and $x_i(p^+)$ or greater than $x_i(p^+)$, for each $1 \le i \le d$. Thus, S_I and S_O can be produced without any additional scalar comparisons by placing each point that has at least one coordinate less than p^- or at least one coordinate greater than p^+ into S_O and all remaining points into S_I. $\square$

Observation 5. *Let $h_0^-, h_1^-, \dots, h_{k+1}^-$ and $h_0^+, h_1^+, \dots, h_{k+1}^+$ be points such that $h_0^- \nearrow h_1^- \nearrow \dots \nearrow h_{k+1}^- \nearrow h_{k+1}^+ \nearrow h_k^+ \nearrow \cdots \nearrow h_0^+$ and consider the regions I and O defined by the pair of points $p^- = h_{k+1}^-$ and $p^+ = h_{k+1}^+$. Each pair of points (h_j^-, h_j^+) defines a hyperrectangle with corner set $\{h_i^\sigma \mid \sigma \in \{+1, -1\}^d\}$ similar to the corner set $\{p^\sigma \mid \sigma \in \{+1, -1\}^d\}$ of I defined by (p^-, p^+). If, for every sign vector $\sigma \in \{+1, -1\}^d$, there exist $k + 1$ points $p_1^\sigma, p_2^\sigma, \dots, p_{k+1}^\sigma$ in S_O such that $h_j^\sigma \nearrow^\sigma p_j^\sigma \nearrow^\sigma h_{j-1}^\sigma$ for all $1 \le j \le k + 1$, then the first k convex layers of S and S_O are identical and p_{k+1}^σ is not part of these layers for any $\sigma \in \{+1, -1\}^d$.*

Lemma 5. *Let S be a point set drawn from a CI distribution, let $0 < \epsilon_1 < \epsilon_2 < 1$ be constants, let $\frac{\tau}{k+1} \ge n^{(\epsilon_2 - 1)/d}$, and let $h_0^-, h_1^-, \dots, h_{k+1}^-, h_0^+, h_1^+, \dots, h_{k+1}^+$ be points such that h_j^- is a $\left(\frac{j}{k+1}\tau, -\mathbf{1}\right)$-pivot and h_j^+ is a $\left(\frac{j}{k+1}\tau, \mathbf{1}\right)$-pivot for all $0 \le j \le k + 1$. Then with probability at least $1 - o\left(n^{-n^{\epsilon_1}}\right)$, every hyperrectangle H_j^σ defined by the points h_{j-1}^σ and h_j^σ, for $1 \le j \le k + 1$ and every sign vector $\sigma \in \{+1, -1\}^d$, contains a point $p_j^\sigma \in S$.*

Proof. Analogous to the proof of Lemma 2, $P(H_j^\sigma \cap S = \emptyset) < e^{-(\tau/(k+1))^d n/4}$, so the probability that there exists a pair (j, σ) such that $H_j^\sigma \cap S = \emptyset$ is less than $(k+1)2^d e^{-(\tau/(k+1))^d n/4}$. As shown in the proof of Lemma 2, $(k+1)e^{-(\tau/(k+1))^d n/4} = o\left(n^{-n^{\epsilon_1}}\right)$. Since d is a constant, this implies that $(k + 1)2^d e^{-(\tau/(k+1))^d n/4} = o\left(n^{-n^{\epsilon_1}}\right)$. $\square$

Lemma 6. *Let S be a point set drawn from a CI distribution, let $0 < \epsilon_1 < \epsilon_2 < 1$ be constants, let $\tau \geq n^{\epsilon_2 - 1}$, let p^- be a $(\tau, -\mathbf{1})$-pivot of S, let p^+ be a $(\tau, \mathbf{1})$-pivot of S, let I be the hyperrectangle defined by (p^-, p^+), let $S_I = S \cap I$, and let $S_O = S \setminus S_I$. Then $E(|S_O|) \leq 2d\tau n$ and $P(|S_O| > 4d\tau n) = o\big(n^{-n^{\epsilon_1}}\big)$.*

Proof. The proof is identical to the proof of Lemma 3 after observing that S_O can be covered with $2d$ halfspaces $B_1^-, B_2^-, \ldots, B_d^-, B_1^+, B_2^+, \ldots, B_d^+$, where $B_i^- = \{p' \in \mathbb{R}^d \mid x_i(p') \leq x_i(p^-)\}$ and $B_i^+ = \{p' \in \mathbb{R}^d \mid x_i(p') \geq x_i(p^+)\}$ for all $1 \leq i \leq d$. □

With these lemmas and observation, we claim that the analysis of the algorithm in Sect. 4 that computes the first k convex layers in two or three dimensions is identical to the proof of Theorem 1 for maximal layers, using Lemmas 4, 5, and 6 and Observation 5 in place of Lemmas 1, 2, and 3 and Observation 4. This completes the proof of Theorem 1.

Convex Layers in Four or Higher Dimensions: We now consider the problem of computing the first k convex layers of a point set S drawn from an arbitrary CI distribution in four or higher dimensions, for $k \leq n^{1/(d^2+2)}$. The framework of Theorem 1 cannot be applied directly to this problem because the best known algorithm for computing even the convex *hull* in $d \geq 4$ dimensions [7] takes $O(n^{\lfloor d/2 \rfloor})$ comparisons.

Bentley et al. [2] showed how to use 2^d skyline computations to produce a superset Q' of the convex hull Q of S of small expected size. Matoušek [13] later called this structure the *quadrant hull* of S. We opt for *orthant hull* here because an orthant is the generalization of quadrants to higher dimensions. We will show later in this section that, with high probability, the size of Q' is small enough so that applying Chazelle's convex hull algorithm to Q' takes $O(n)$ comparisons. Combined with Kung et al.'s algorithm for computing the skyline in d dimensions in $O(n \log^{d-2} n)$ time [12], this gives an algorithm M that computes the convex hull of S using $O(n \log^{d-2} n)$ comparisons with high probability. To compute $k > 1$ convex layers, M repeats this process k times: the ith iteration computes the convex hull of the point set left after removing the first $i - 1$ convex layers. With high probability, this will take $O(kn \log^{d-2} n)$ time because, as we show below, the size not only of the orthant hull but in fact of the first k *orthant layers* is small enough to apply Chazelle's algorithm k times.

To prove Theorem 2, we use M in conjunction with Theorem 1 where $c = 1$. Theorem 1 requires M to achieve a running time of $O(kn^{1+\epsilon})$ in the worst case. However, it is easily verified that the proof of Theorem 1 continues to hold if M achieves this running time with some probability $p > 0$, in which case Theorem 1 produces a convex hull algorithm that uses $2dn + o(n)$ scalar comparisons with probability $\Theta\Big(\min\Big(p, 1 - o\Big(n^{-n^{\epsilon'}}\Big)\Big)\Big)$. Since we prove below that M achieves a running time of $O(n \log^{d-2} n)$ with probability $1 - O\Big(\frac{1}{n^{1/d - \epsilon}}\Big)$, Theorem 2 follows by setting $\epsilon = 1/(2d) - 1/(d^2+2)$ in Theorem 1.

Let $\sigma \in \{+1, -1\}^d$ be a sign vector, let $\sigma \circ S = \{\sigma \circ p \mid p \in S\}$, and let L^σ be the set of points $p \in S$ such that $\sigma \circ p$ belongs to the skyline of $\sigma \circ S$.

We call L^σ the *σ-skyline* of S. The *orthant hull* of S is $Q' = \bigcup_{\sigma \in \{+1,-1\}^d} L^\sigma$, and Bentley et al. proved that $Q \subseteq Q'$. To define the *orthant layers* of S, let Q' be the first orthant layer of S and, for $i > 1$, let the ith orthant layer of S be the orthant hull of the subset of S obtained after removing the first $i-1$ orthant layers from S.

Let $Q_1, Q_2, \ldots, Q_k$ be the first k convex layers of S, let $S_i = S \setminus \bigcup_{j=1}^{i-1} Q_j$, and let Q'_i be the orthant hull of S_i for all $1 \le i \le k$. Since Q_i is the convex hull of S_i, Bentley et al.'s result shows that $Q_i \subseteq Q'_i$ and it is not hard to see that $\bigcup_{i=1}^k Q'_i$ is a subset of the points on the first k orthant layers[1] of S. Computing Q'_i in the ith iteration takes $O(2^d n \log^{d-2} n) = O(n \log^{d-2} n)$ time by applying Kung et al.'s algorithm once for each sign vector σ. Summing over all k iterations, we get an upper bound $O(kn \log^{d-2} n)$. To compute Q_i, we apply Chazelle's algorithm to Q'_i, which takes $O(|Q'_i|^{\lfloor d/2 \rfloor})$ time. Summing over all k layers, we obtain that computing the first k convex layers using M takes $O(kn \log^{d-2} n + \sum_{i=1}^k |Q'_i|^{\lfloor d/2 \rfloor}) = O(kn \log^{d-2} n + k|Q''|^{\lfloor d/2 \rfloor})$ time, where Q'' is the set of points on the first k orthant layers of S. As we show in Sect. 5.2, $E(k^{2/d}|Q''|) = O(k^{2/d} k^d \log^{d-1} n) = O(n^{1/d} \log^{d-1} n)$ because $k \le n^{1/(d^2+2)}$. Thus, by Markov's inequality, $P(k^{2/d}|Q''| > n^{2/d}) \le n^{-1/d+\epsilon}$, that is, M takes $O(kn \log^{d-2} n)$ time with probability at least $1 - n^{-1/d+\epsilon}$, as claimed.

5 Expected Size of the First *k* Layers

Bentley et al. [2] proved that the expected size of the skyline of a point set drawn from a CI distribution in d dimensions is $O(\log^{d-1} n)$. They also used this result to give a bound of $O(\log^{d-1} n)$ on the expected number of vertices on the convex hull. It seems difficult to extend their technique to subsequent layers. In Sect. 5.1, we show that, for continuous CI distributions, the kth maximal layer has expected size $O(k^d \log^{d-1}(n/k^d))$. The proof is based on a proof sketch for 2D suggested by an anonymous reviewer of an earlier draft of this paper. In Sect. 5.2, we show how to extend the argument to obtain the same bound (up to a factor of 4^d) for convex and orthant layers. This proves Theorem 3.

5.1 Maximal Layers

First consider a point set S drawn uniformly at random from the unit hypercube. To simplify the discussion, we bound the size of the kth *minimal* layer of S, which is equivalent to the kth maximal layer via the transformation $(x_1, x_2, \ldots, x_d) \mapsto (1-x_1, 1-x_2, \ldots, 1-x_d)$. For every point $p \in \mathbb{R}^d$, let D_p be the set of points dominated by p, and let $|D_p|$ be the volume of D_p. For every integer $t \ge 0$, let B_t be the set of all points $p \in [0,1]^d$ with $|D_p| = \frac{(2k)^d t}{n}$, that is, a point $(x_1, x_2, \ldots, x_d)$ belongs to B_t if and only if $x_1 x_2 \cdots x_d = \frac{(2k)^d t}{n}$ and $0 \le x_i \le 1$

[1] Note that $\bigcup_{i=1}^k Q'_i$ is not a subset of the first k *σ-skyline* of S. A counterexample will be given in the full version of this paper.

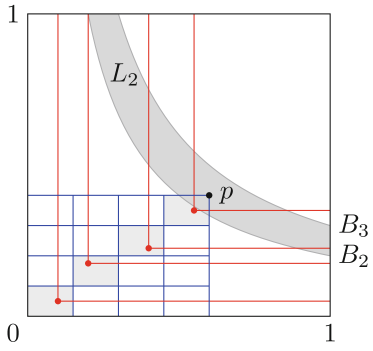


Fig. 2. Bound on the probability that a point in L_t belongs to the first k minimal layers in 2D. Here, $k = 2$ and $n = 160$. The region L_2 is shaded. Each of the grey grid cells contains two points of S in expectation. Any point in such a grid cell (red) is dominated by p and all points in grid cells to its top right. Thus, unless more than two of these grid cells are empty, p does not belong to the first two minimal layers. (Color figure online)

for all $1 \le i \le d$. B_t splits the unit hypercube into two regions: L_t^- includes $(0,\dots,0)$ and L_t^+ includes $(1,\dots,1)$. For $t \ge 0$, let $L_t = L_t^+ \cap L_{t+1}^-$ be the region between by B_t and B_{t+1}. See Fig. 2 for an example. The volume of L_t^- is bounded by

$$\int_{(2k)^d t/n}^{1} \int_{(2k)^d t/n}^{1} \cdots \int_{(2k)^d t/n}^{1} \frac{(2k)^d t}{n x_1 x_2 \cdots x_{d-1}} dx_1 dx_2 \cdots dx_{d-1} + \frac{d(2k)^d t}{n}$$
$$= O\left(\frac{k^d t \log^{d-1}(n/k^d)}{n}\right).$$

Since $L_t \subseteq L_{t+1}^-$, this implies that $|L_t| = O\Big(\frac{k^d t \log^{d-1}(n/k^d)}{n}\Big)$. Next consider a point $p \in L_t$ and divide D_p into a uniform grid with $(2k)^d$ cells by dividing each side of D_p into $2k$ equal intervals. Each cell of this grid has volume at least t/n and thus contains at least t points in expectation. Thus, using the Chernoff bound, any of these grid cells is empty with probability less than $e^{-t/4}$. For p to be on one of the first k minimal layers, at least k of the $2k$ cells on the diagonal of the grid must be empty, which happens with probability less than $2e^{-t/4}$, by Markov's inequality. Thus, any point in S belongs to L_t and to one of the first k layers with probability $O(\frac{k^d t \log^{d-1}(n/k^d)}{n e^{t/4}})$. The expected number of points in S that belong to L_t and to one of the first k layers is thus $O(\frac{k^d t \log^{d-1}(n/k^d)}{e^{t/4}})$. Since $\sum_{t=0}^{\infty} \frac{t}{e^{t/4}} = O(1)$, the expected number of points on the first k minimal layers is thus $O(k^d \log^{d-1}(n/k^d))$.

For an arbitrary continuous CI distribution $\mathcal{D}$, let P_i be the cumulative distribution function of the probability distribution of the ith coordinate. Then, for any point set S drawn from $\mathcal{D}$, the mapping $\phi : (x_1, x_2, \ldots, x_d) \mapsto (P_1(x_1), P_2(x_2), \ldots, P_d(x_d))$ produces a point set S' drawn uniformly at random from the unit hypercube and $p \in S$ dominates $q \in S$ if and only if $\phi(p)$ dominates $\phi(q)$. Thus, the total expected size of the first k maximal layers, and thus the expected size of the kth maximal layer, of S is $O(k^d \log^{d-1}(n/k^d))$.

5.2 Convex Layers

By the argument in the previous paragraph, it suffices to prove that the first k convex layers of a point set S drawn uniformly at random from the unit hypercube have expected size $O(k^d \log^{d-1}(n/k^d))$. Let o be the center point of the unit hypercube. Point o splits the unit hypercube into 2^d orthants $\mathcal{O}^\sigma$ with side length $1/2$ for $\sigma \in \{+1, -1\}^d$. More precisely, $\mathcal{O}^\sigma$ is the set of all points in the unit hypercube that σ-dominate o. We prove that the expected number of points in each orthant $\mathcal{O}^\sigma$ that belong to the first k convex layers is $O((2k)^d \log^{d-1}(n/k^d))$. Summing over all 2^d orthants, we obtain a bound of $O((4k)^d \log^{d-1}(n/k^d)) = O(k^d \log^{d-1}(n/k^d))$ on the expected size of the first k convex layers of S. W.l.o.g. consider the orthant $\mathcal{O}^{-\mathbf{1}}$; the argument for any other orthant $\mathcal{O}^\sigma$ is analogous after negating the point coordinates of S in all dimensions where σ and $-\mathbf{1}$ differ.

We define sets $L_1, L_2, \ldots$ as in Sect. 5.1. The key of our proof is to show that any point in $L_t \cap \mathcal{O}^{-\mathbf{1}}$ belongs to one of the first k convex layers with probability at most $2^{d+1}e^{-t/4}$. Since $L_t \cap \mathcal{O}^{-\mathbf{1}} \subseteq L_t$, the exact same calculation as in Sect. 5.1 then shows that the expected number of points in $L_t \cap \mathcal{O}^{-\mathbf{1}}$ that belong to the first k convex layers is $O\Bigl(\frac{(2k)^d t \log^{d-1}(n/k^d)}{e^{t/4}}\Bigr)$ and again, since $\sum_{t=0}^{\infty} \frac{t}{e^{t/4}} = O(1)$, the expected number of points in $\mathcal{O}^{-\mathbf{1}}$ that belong to the first k convex layers is $O((2k)^d \log^{d-1}(n/k^d))$, as claimed.

So consider a point $p \in L_t \cap \mathcal{O}^{-\mathbf{1}}$. Let D_p^σ be the part of the unit hypercube σ-dominated by p. We divide each hyperrectangle D_p^σ into a uniform grid with $(2k)^d$ cells by dividing each side of D_p^σ into $2k$ equal intervals. Consider the diagonal of D_p^σ connecting p with the opposite corner of D_p^σ and let $H_1^\sigma, H_2^\sigma, \ldots, H_{2k}^\sigma$ be the grid cells intersected by this diagonal, ordered by increasing distance from p. The argument in Sect. 5.1 shows that $P(S \cap H_i^{\mathbf{1}} = \emptyset) < e^{-t/4}$ for all $1 \le i \le 2k$, since $D_p = D_p^{\mathbf{1}}$. Since $p \in \mathcal{O}^{-\mathbf{1}}$, we have $|H_i^\sigma| \ge |H_i^{\mathbf{1}}|$ for all i and all σ. Thus, $P(S \cap H_i^\sigma = \emptyset) \le P(S \cap H_i^{\mathbf{1}}) < e^{-t/4}$. Now, if there exist k indices $1 \le i_1 < i_2 < \cdots < i_k \le 2k$ such that, for all $1 \le j \le k$ and all $\sigma \in \{-1, +1\}^d$, $H_{i_j}^\sigma \cap S \ne \emptyset$, then p is not on the kth convex layer. Thus, for p to be on the kth convex layer, there have to be at least k indices $1 \le i'_1 < i'_2 < \cdots < i'_k \le 2k$ and sign vectors $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that $S \cap H_{i'_j}^{\sigma_j} = \emptyset$ for all $1 \le j \le k$. For any fixed index $1 \le i \le 2k$, the probability that there exists a sign vector σ_i such that $S \cap H_i^{\sigma_i} = \emptyset$ is less than $2^d e^{-t/4}$, since $P(S \cap H_i^\sigma = \emptyset) < e^{-t/4}$ for any fixed i and σ. Thus, the expected number of indices i such that $S \cap H_i^{\sigma_i} = \emptyset$ for some sign vector σ_i is less than $2^{d+1}ke^{-t/4}$. By Markov's inequality, the probability that there are at least k such indices is thus less than $2^{d+1}e^{-t/4}$. Since this is an upper bound on the probability that p belongs to the first k convex layers, this finishes the proof of Theorem 3 for convex layers.

To obtain the same bound for the expected size of the first k orthant layers, observe that point p does not belong to the first k orthant layers if there exist k indices $1 \leq i_1 < i_2 < \cdots < i_k \leq 2k$ such that, for all $1 \leq j \leq k$ and all $\sigma \in \{-1, +1\}^d$, $H^{\sigma}_{i_j} \cap S \neq \emptyset$. Since this is the same condition we used to bound the size of the first k convex layers, the above argument also shows that the first k orthant layers of S have expected size $O(k^d \log^{d-1}(n/k^d))$.

References

1. Bentley, J.L., Clarkson, K.L., Levine, D.B.: Fast linear expected-time algorithms for computing maxima and convex hulls. Algorithmica **9**(2), 168–183 (1993)
2. Bentley, J.L., Kung, H.T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. J. ACM **25**(4), 536–543 (1978)
3. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
4. Blunck, H., Vahrenhold, J.: In-place algorithms for computing (layers of) maxima. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 363–374. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_34
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, pp. 421–430 (2001)
6. Buchsbaum, A.L., Goodrich, M.T.: Three-dimensional layers of maxima. Algorithmica **39**(4), 275–286 (2004)
7. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discret. Comput. Geom. **10**(4), 377–409 (1993)
8. Chazelle, B.: On the convex layers of a planar set. IEEE Trans. Inf. Theor. **31**(4), 509–517 (2006)
9. Dalal, K.: Counting the onion. Random Struct. Algorithms **24**(2), 155–165 (2004)
10. Frieze, A.: On the length of the longest monotone subsequence in a random permutation. Ann. Appl. Probab. **1**(2), 301–305 (1991)
11. Golin, M.J.: A provably fast linear-expected-time maxima-finding algorithm. Algorithmica **11**(6), 501–524 (1994)
12. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975)
13. Matoušek, J., Plecháč, P.: On functional separately convex hulls. Discret. Comput. Geom. **19**(1), 105–130 (1998)
14. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
15. Nielsen, F.: Output-sensitive peeling of convex and maximal layers. Inf. Process. Lett. **59**(5), 255–259 (1996)
16. Okabe, A., Boots, B., Sugihara, K., Chiu, S., Kendall, D.G.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley, Hoboken (2008)

Plane Gossip: Approximating Rumor Spread in Planar Graphs

Jennifer Iglesias[1](✉), Rajmohan Rajaraman[2], R. Ravi[1], and Ravi Sundaram[2]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
{jiglesia,ravi}@andrew.cmu.edu
[2] Northeastern University, Boston, MA, USA
{rraj,koods}@ccs.neu.edu

Abstract. We study the design of schedules for multi-commodity multicast. In this problem, we are given an undirected graph G and a collection of source-destination pairs, and the goal is to schedule a minimum-length sequence of matchings that connects every source with its respective destination. The primary communication constraint of the multi-commodity multicast model is the number of connections that a given node can make, not link bandwidth. Multi-commodity multicast and its special cases, (single-commodity) broadcast and multicast, are all NP-complete. Multi-commodity multicast is closely related to the problem of finding a subgraph of optimal poise, where the poise is defined as the sum of the maximum degree and the maximum distance between any source-destination pair. We show that for any instance of the multicast problem, the minimum poise subgraph can be approximated to within a factor of $O(\log k)$ with respect to the value of a natural LP relaxation in a graph with k source-destination pairs. This is the first upper bound on the integrality gap of the natural LP; all previous algorithms yielded approximations with respect to the integer optimum. Using this integrality gap upper bound and shortest-path separators in planar graphs, we obtain our main result: an $O(\log^3 k \frac{\log n}{\log\log n})$-approximation for multi-commodity multicast for planar graphs which improves on the $2^{\widetilde{O}(\sqrt{\log n})}$-approximation for general graphs.

We also study the minimum-time radio gossip problem in planar graphs where a message from each node must be transmitted to all other nodes under a model where nodes can broadcast to all neighbors and only nodes with a single broadcasting neighbor get a non-interfered message. In earlier work Iglesias et al. (FSTTCS 2015), we showed a strong $\Omega(n^{\frac{1}{2}-\epsilon})$-hardness of approximation for computing a minimum gossip schedule in general graphs. Using our techniques for the telephone model, we give an $O(\log^2 n)$-approximation for radio gossip in planar graphs breaking this barrier. Moreover, this is the first bound for radio gossip given that doesn't rely on the maximum degree of the graph.

This material is based upon research supported in part by the U. S. Office of Naval Research under award number N00014-12-1-1001 and National Science Foundation under award number CCF-1527032.

J. Iglesias—Now at Waymo. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2013170941.

1 Introduction

Rumor spreading in networks has been an active research area with questions ranging from finding the minimum possible number of messages to spread gossip around the network [3,17,33] to finding graphs with minimum number of edges that are able to spread rumors in the minimum possible time in the network [16]. There is also considerable work in the distributed computing literature on protocols for rumor spreading and gossip based on simple push and pull paradigms (e.g., see [10,11,15,20]).

The focus of this paper is the class of problems seeking to minimize the time to complete the rumor spread, the prototypical example being the **minimum broadcast time problem** where a message at a root node must be sent to all nodes via connections represented by an undirected graph in the minimum number of rounds. Under the popular "telephone" model, every node can participate in a telephone call with at most one other neighbor in each round to transmit the message, and the goal is to minimize the number of rounds. This problem has seen active work in designing approximation algorithms [4,9,22,30]. One generalization of broadcast is the **minimum multicast time problem**: We are given an undirected graph $G = (V, E)$ representing a telephone network on V, where two adjacent nodes can place a telephone call to each other. We are given a source vertex r and a set of terminals $R \subseteq V$. The source vertex has a message and it wants to inform all the terminals of the message. To do this, the vertices of the graph can communicate in rounds using the telephone model. The goal is to deliver the message to all terminals in the minimum number of rounds.

Recently, a more general demand model called the multicommodity multicast was introduced in [28]. In the **minimum multicommodity multicast time problem**, a graph $G(V, E)$ is given along with a set of pairs of nodes $P = \{(s_i, t_i) | 1 \leq i \leq k\}$, known as demand pairs. Each vertex s_i has a message m_i which needs be delivered to t_i. The vertices communicate similar to the multicast problem. The goal is to deliver the message from each source to its corresponding sink in the minimum number of rounds. Note that there is no bound on the number of messages that can be exchanged in a telephone call. In this sense, the telephone model captures a classic information dissemination problem where the primary communication constraint is the number of connections that a given node can make in each round, not link bandwidth.

1.1 Poly-logarithmic Approximation for Planar Multicommodity Multicast

While even sub-logarithmic ratio approximations have been known for the minimum time multicast problem [4,9,22,30], the best known approximation guarantees for the multicommodity case [28] is $\tilde{O}(2^{\sqrt{\log k}})$ where k is the number of different source-sink pairs.

Theorem 1. *There is a polynomial time algorithm for minimum time multicommodity multicast with k source-sink pairs in a n-node undirected planar graphs that constructs a schedule of length $O(OPT \log^3 k \frac{\log n}{\log\log n})$ where OPT is the length of the optimal schedule.*

This result extends in a natural way to bounded genus graphs. Our results make critical use of the fact that planar graphs admit small-size balanced vertex separators that are a combination of three shortest paths starting from any given node [32]. We aggregate messages at the paths, move them along the path and then move them onto their destinations using a local multicast. To break the overall multi-commodity multicast problem into recursive subproblems, we solve an LP relaxation for the overall problem and for those pairs for which the LP uses the separator path nodes in sending messages by a "large" amount, we aggregate them to the separator paths and move them along the paths. However, to define this aggregation automatically we need to use a linear program which requires us to relate another lower bound for the schedule length that we describe next.

1.2 Poise and a New LP Rounding Algorithm

Suppose that the (single-commodity) multicast problem in a graph G with root r and terminals R admits a multicast schedule of length L. Consider all the nodes $I \subseteq V$ in the graph that are informed of the message from the root in the course of the schedule. For every node $v \in I$ consider the edge through which v first heard the message and direct this edge into v. It is easy to verify that this set of arcs forms an out-arborescence T rooted at r and spanning I. In particular, every node in I except r has in-degree exactly one and there is a directed path from r to every vertex in I.

Definition 1. *Define the poise of an undirected tree T to be the sum of the diameter of the tree and the maximum degree of any node in it. Define the poise of a directed tree to be that of its undirected version (ignoring directions).*

The discussion above of constructing a directed tree from a multicast schedule implies that the poise of the tree constructed from a multicast schedule of length L is at most 3L (see also [30]). The following lemma gives the relation in the other direction.

Lemma 1. [30] *Given a tree on n nodes of poise L, there is a polynomial time algorithm to construct a broadcast scheme of length $O(L \cdot \frac{\log n}{\log\log n})$ from any root.*

Note that a complete d-ary regular tree of depth d requires time d^2 to finish multicast from the root; If the size of the tree is n, then $d = O(\frac{\log n}{\log\log n})$. For this tree $L = O(\frac{\log n}{\log\log n})$ while any broadcast scheme takes $\Omega((\frac{\log n}{\log\log n})^2)$ steps showing that the multiplicative factor is necessary.

Even though approximation algorithms for minimum poise trees connecting a root to a set of terminals were known from earlier work [4,9,30], their guarantees are with respect to an optimal (integral) solution and not any specific LP relaxation. In particular, the LP-based algorithm of [30] rounds a solution to the poise LP in phases without preserving the relation of the residual LPs that arise in the phases to the LP for the poise of the whole graph. Similarly, the LP-based algorithm of [4] solves a series of LPs determining how to hierarchically pair terminals and form the desired broadcast tree with cost within a logarithmic factor of the integral optimum poise, but without relating the resulting tree to the LP value of the poise of the original graph. It is not straightforward to use these methods to derive an integrality gap for the minimum poise LP, and this has remained an open problem. Deriving an approximation algorithm for minimum poise subgraphs for the single-commodity multicast version with a small *integrality gap* is a critical ingredient in our approximation algorithm for multicommodity multicast problem in planar graphs (Theorem 1). We derive the first such result.

Theorem 2. *Given a fractional feasible solution of value L to a natural linear programming relaxation of the minimum poise of a tree connecting a root r to terminals R (POISE-L LP, see Sect. 2), there is a polynomial time algorithm to construct a tree spanning $r \cup R$ of poise $O(L \log k)$ where $k = |R|$ and $n = |V|$.*

Our LP rounding for minimum poise are based on exploiting a connection to the theory of multiflows [5,12,25]; this is an interesting technique in its own right that we hope will be useful in obtaining other LP rounding results for connectivity structures while preserving degrees and distances.

1.3 Radio Gossip in Planar Graphs

Our techniques for addressing multicommodity multicast are also applicable to radio gossip in planar graphs. In the radio model, communication also occurs in rounds; a transmitting node may broadcast to multiple nodes in a round but a node may receive successfully in a given time step only if exactly one of its neighbors transmits. The gossip problem is a special case of the multicommodity multicast problem where the demand pairs include all possible pairs of nodes (alternately, every node's message must be transmitted to every other node). The minimum gossip problem in the radio model has been widely studied [14], but all known upper bounds involve both the diameter and degree of the network. In particular, for general n-node graphs, there is an $\Omega(n^{\frac{1}{2}-\epsilon})$-hardness of approximation result for computing a minimum gossip schedule [18]. Our next result breaks this barrier for planar graphs.

Theorem 3. *There is a polynomial time algorithm for minimum time radio gossip in an n-node undirected planar graph that constructs a schedule of length $O(OPT \cdot \log^2 n)$ where OPT is the length of the optimal gossip schedule.*

Since radio broadcast from any node can already be achieved with additive poly-logarithmic time overhead above the optimum [26], our algorithm for radio gossip focuses on gathering all the messages to a single node. For this, we use the path-separator decomposition in planar graphs to recursively decompose the graph and gather messages bottom up. However, the diameter of subgraphs formed by the decomposition are not guaranteed to be bounded so we use a carefully constructed degree-bounded matching subproblem to accomplish the recursive gathering: these techniques adapt and extend the methods used for constructing telephone multicast schedules [28] but apply them for the first time to the radio gathering case.

Both our results on planar graphs also naturally extend to minor-free graphs; details on the extentsion are in the full version of our paper [19].

1.4 Previous Work

Minimum time multicast in the telephone model. Finding optimal broadcast schedules for trees was one of the first theoretical problems in this setting and was solved using dynamic programming [29]. For general graphs, Kortsarz and Peleg [22] developed an additive approximation algorithm which uses at most $c \cdot OPT + O(\sqrt{n})$ rounds for some constant c in an n-node graph. They also present algorithms for graphs with small balanced vertex separators with approximation ratio $O(\log n \cdot S(n))$ where $S(n)$ is the size of the minimum balanced separator on graphs of size n from the class. The first poly-logarithmic approximation for minimum broadcast time was achieved by Ravi [30] and the current best known approximation ratio is $O(\frac{\log n}{\log\log n})$ due to Elkin and Korsartz [9]. The best known lower bound on the approximation ratio for telephone broadcast is $3 - \epsilon$ [6].

In his study of the telephone broadcast problem, Ravi [30] introduced the idea of finding low poise spanning trees to accomplish broadcast. In the course of deriving a poly-logarithmic approximation, Ravi also showed how a tree of poise P in an n-node graph can be used to complete broadcast starting from any node in $O(P \cdot \frac{\log n}{\log\log n})$ steps. His result provided an approximation guarantee with respect to the optimal poise of a tree but not its natural LP relaxation that we investigate.

Guha et al. [4] improved the approximation factor for multicasting in general graphs to $O(\log k)$ where k is the number of terminals. The best known approximation factor for the multicast problem is $O(\frac{\log k}{\log\log k})$ [9]. Both of [4,9] present a recursive algorithm which reduces the total number of uninformed terminals in each step of the recursion, while using $O(OPT)$ number of rounds in that step. In [4], they reduce the number of uninformed terminals by a constant factor in each step and so they obtain a $O(\log k)$-approximation, but in [9], the number of uninformed terminals is reduced by a factor of OPT which gives a $O(\frac{\log k}{\log\log k})$-approximation due to the fact that $OPT = \Omega(\log k)$. These papers also imply an approximation algorithm with factors $O(\log k)$ and $O(\frac{\log k}{\log\log k})$ for the Steiner minimum poise subgraph problem; however, these guarantees are again with respect to the optimum integral value for this problem and not any fractional relaxation.

For the multicommodity multicast problem, Nikzad and Ravi [28] adapt the methods of [8,9] to present an algorithm with approximation ratio $\tilde{O}(2^{\sqrt{\log k}})$

where k is the number of different source-sink pairs. They also show that there is a poly-logarithmic approximation inter-reducibility between the problem of finding a minimum multicommodity multicast schedule and that of finding a subgraph of minimum generalized Steiner poise (i.e., a subgraph that connect all source-sink pairs, but is not necessarily connected overall, and has minimum sum of maximum degree and maximum distance in the subgraph between any source-sink pair).

Radio Gossip. The radio broadcast and gossip problems have been extensively studied (see the work reviewed in the survey [13]). The best-known scheme for radio broadcast is by Kowalski and Pelc [23] which completes in time $O(D + \log^2 n)$, where n is the number of nodes, and D is the diameter of the graph and is a lower bound to get the message across the graph from any root. The $O(\log^2 n)$ term is also unavoidable as demonstrated by Alon et al. [2] in an example with constant diameter that takes $\Omega(\log^2 n)$ rounds for an optimal broadcast scheme to complete. Elkin and Korsartz [7] also show that achieving a bound better than additive log-squared is not possible unless $NP \subseteq DTIME(n^{\log\log n})$. For planar graphs, the best upper bound for radio broadcast time is $D + O(\log n)$ given by [26]. The best bound for radio gossip known so far, however, is $O(D + \Delta \log n)$ steps in an n-node graph with diameter D and maximum degree Δ [14], even though there is no relation in general between the optimum radio gossip time and the maximum degree. Indeed, for general graphs, there is a polynomial inapproximability lower-bound for the minimum time radio gossip problem [18].

Planar path separators. For our results on planar graphs, we rely on the structure of path-separators. Lipton and Tarjan first found small $O(\sqrt{n})$-sized separators for n-node undirected planar graphs [24]. More recently, planar separators based on any spanning tree of a planar graph were found [32] with the following key property: these balanced vertex separators can be formed by starting at any vertex and taking the union of three shortest paths from this vertex. Minor-free graphs also admit small path-separators as found by [1]; in this case, the number of paths used depends on the graphs which are excluded minors, but stays constant for constant-sized excluded minors.

2 LP Rounding for Multicast in General Graphs

In this section we present an approximation algorithm for finding a **minimum poise Steiner subgraph**, and establish an LP integrality gap upper bound, thus proving Theorem 2. We begin by presenting a linear program for a multicommodity generalization of minimum poise Steiner subgraph, which is useful for the multicommodity multicast problem. This linear program, when specialized to the case where we need to connect a root r to a subset R of terminals, is our LP for the minimum poise Steiner subgraph problem.

2.1 Linear Program for Poise

The generalized Steiner poise problem is to determine the existence of a subgraph containing paths for every demand pair in $K = \{(s_i, t_i) | 1 \leq i \leq k\}$ of poise at most L, i.e. every demand pair is connected by a path of length at most L and every node in the subgraph has degree at most L.

We use indicator variables $x(e)$ to denote the inclusion of edge e in the subgraph. Since the poise is at most L, this is also an upper bound on the length of the path from any terminal to the root. For every terminal $(s_i, t_i) \in K$, define $\mathcal{P}_i$ to be the set of all (simple) paths from s_i to t_i. We use a variable $y_t(P)$ for each path $P \in \mathcal{P}_i$ that indicates whether this is the path used by s_i to reach t_i in the subgraph. For a path P, let $\ell(P)$ denote the number of hops in P. The integer linear program for finding a subgraph of minimum poise is given below.

$$\begin{array}{lll}
\text{minimize} & L = L_1 + L_2 & (POISE - L) \\
\text{subject to} & \sum_{e \in \delta(v)} x(e) \leq L_1 & \forall v \in V \\
& \sum_{P \in \mathcal{P}(i)} y_i(P) = 1 & \forall i \in K \\
& \sum_{P \in \mathcal{P}(i)} \ell(P) y_i(P) \leq L_2 & \forall i \in K \\
& \sum_{P \in \mathcal{P}(i): e \in P} y_i(P) \leq x(e) & \forall e \in E, i \in K \\
& x(e) \in \{0,1\} \text{ for } e \in E, & \\
& y_i(P) \in \{0,1\} \text{ for } i \in K, P \in \mathcal{P}_L(i). &
\end{array}$$

The first set of constraints specifies that the maximum degree of any node using the edges in the subgraph is at most L_1. The second set insists that there is exactly one path chosen between every pair $(s_i, t_i) \in K$. The third set ensures that the length of the path thus selected is at most L_2. The fourth set requires that if the path $P \in \mathcal{P}_i$ is chosen to connect s_i to t_i, all the edges in the path must be included in the subgraph.

We will solve the LP obtained by relaxing the integrality constraints to non-negativity constraints[1], and get an optimal solution $x, y \geq 0$.

For the remainder of this section, we will focus on the rooted version of this problem. In particular, there will be a root r and set of terminals R, then we will make $K = \{(r,t) | t \in R\}$. It still remains to round a solution to POISE-LP to prove Theorem 2. Before presenting the rounding algorithm in Sect. 2.3, we describe a result on multiflows that will be useful in decomposing our LP solution into a set of paths that match terminals with each other.

2.2 Preliminaries

Given an undirected multigraph G with terminal set $T \subset V$ of nodes, a *multiflow* is an edge-disjoint collection of paths each of which start and end in two distinct terminals in T. The value of the multiflow is the number of paths in the collection. Such a path between two distinct terminals is called a T-path and a multiflow is called a T-path packing. For any terminal $t \in T$, let $\lambda(t, T \setminus t)$ denote the minimum cardinality of an edge cut separating t from $T \setminus t$ in G. Note that in any multiflow, the maximum number of paths with t as an endpoint is at most

[1] Even though the number of path variables is exponential, it is not hard to convert this to a compact formulation on the edge variables that can be solved in polynomial time. See e.g., [30].

$\lambda(t, T \setminus t)$. Furthermore, since every path in a multiflow has to end in distinct vertices in T, the maximum value of any multiflow for T is upper bounded by $\sum_{t \in T} \frac{\lambda(t, T \setminus t)}{2}$, by summing over the maximum number of possible paths from each terminal and dividing by two to compensate for counting each path from both sides. This upper bound can be achieved if a simple condition is met.

Theorem 4. [5,25] *If every vertex in $V \setminus T$ has even degree, then there exists a multiflow for T of value $\sum_{t \in T} \frac{\lambda(t, T \setminus t)}{2}$.*

The following simple construction will be useful in the rounding algorithm to identify good paths to merge clusters. It is based on a lemma from [30], and is in the full version [19].

Lemma 2. *Let G be a digraph where every node has at most one outgoing edge (and no self loops). In polynomial time, one can find an edge-induced subgraph H of G such that H is a partition of the nodes of G into a forest of directed trees each being an inward arborescence, and with $|E(H)| \geq |E(G)|/2$.*

2.3 The Rounding Algorithm

The main idea of Algorithm 1 is to work in $O(\log k)$ phases, reducing the number of terminal-containing components in the subgraph being built by a constant fraction at each stage [31]. We begin with an empty tree containing only the terminals R, each in a cluster by themselves. In each phase, we will merge a constant fraction of the clusters together carefully so that the diameter of any cluster increases by at most an additive $O(L)$ per phase: for this, we choose a terminal as a center of each cluster. When we merge clusters, we partition the clusters into stars where we have paths of length $O(L)$ from the centers of the star leaf clusters to the center of the star center-cluster. These steps closely follow those in [30]. The crux of the new analysis is to extract a set of stars that merge a constant fraction of the current cluster centers using a solution to POISE-L LP.

The key subroutine to determine paths to merge centers is presented in Algorithm 2. This uses the multiflow packing theorem of [5,25].

Algorithm 1. LP Rounding for Poise-L tree

```
1: Clusters $\mathcal{C} \leftarrow R$; Centers $\mathcal{C}^* \leftarrow R$; Solution graph $H \leftarrow \emptyset$; Iteration $i \leftarrow 1$.
2: while $|\mathcal{C}| > 1$ do
3:    Use Algorithm Merge-Centers($\mathcal{C}^*$) to identify a subgraph $F_i$ whose addition
      reduce the number of clusters by a constant fraction;
4:    $H \leftarrow H \cup F_i$; Update $\mathcal{C}$ to be the set of clusters after adding the subgraph $F_i$, and
      update $\mathcal{C}^*$ to be the centers of the updated clusters based on the star structure
      from Algorithm Merge-Centers($\mathcal{C}^*$). Increment $i$.
5: end while
6: Add a path of length at most $L$ from $r$ to the center of the final cluster in $H$. Find
   a shortest path tree in $H$ rooted at $r$ reaching all the terminals in $R$ and output it.
```

Algorithm 2. Merge-Centers($\mathcal{C}^*$) using LP solution x

```
1: Multiply the POISE-LP solution $x$ by the least common multiple $M$ of the denominators in the nonzero values of $x$ to get a multigraph.
2: For every terminal $t \in \mathcal{C}^*$, retain the edges in the paths corresponding to the paths in its LP-solution with nonzero value (i.e., paths $P$ with nonzero $y_t(P)$), for a total of $M$ connectivity from $t$ to $r$. Note that the union of all the retained edges gives connectivity $M$ from every $t \in \mathcal{C}^*$ to $r$ and hence by transitivity, between each other.
3: Double each edge in the multigraph to make it even degree, and use Theorem 4 to find a multiflow of value $\sum_{t\in\mathcal{C}^*} \frac{\lambda(t,\mathcal{C}^*\setminus t)}{2} \geq \sum_{t\in\mathcal{C}^*} \frac{2M}{2} = |\mathcal{C}^*| \cdot M$. Note that each terminal in $\mathcal{C}^*$ has at least $M$ paths in the multiflow.
4: For every terminal $t$, pick one of the $M$ paths incident on it uniformly at random and set this path to be $P_t$. If the chosen path has length longer than $4L$, then eliminate it from further consideration and set $P_t \leftarrow \emptyset$.
5: Let $H$ be an auxiliary graph on vertex set $\mathcal{C}^*$ with at most one arc coming out of each $t \in \mathcal{C}^*$ pointing to the other endpoint of $P_t$ (or add no edge if $P_t = \emptyset$).
6: Apply Lemma 2 to the subgraph of $H$ made of nodes, to get a collection $H'$ of in-trees. For each in-tree, partition the arcs into those in odd and even levels of the tree and pick the set with the larger number of arcs. Note that these sets form stars originating from a set of centers and going to a single center. Let $H''$ denote the set of these stars.
7: For each arc of the stars in $H''$, include the path $P_t$ originating at the leaf of the star corresponding to the arc in $H''$, and output the collection of paths.
```

2.4 Performance Ratio

In this section, we prove Theorem 2. The performance ratio of the rounding algorithm in the theorem is a consequence of the following claims, the first of which follows directly from the path pruning in Algorithm 2. The proof of the second is left to the full version [19].

Lemma 3. *The length of each path output by Merge-Centers($\mathcal{C}^*$) is at most $4L$.*

Lemma 4. *The expected number of paths output by Merge-Centers($\mathcal{C}^*$) is $\Omega(|\mathcal{C}^*|)$.*

Lemma 5. *The distance of any node in a cluster to its center increases by at most $4L$ in the newly formed cluster by merging paths corresponding to stars in H''. Thus, the diameter of any cluster in iteration i is at most $8iL$.*

Proof. The proof is by induction over i, and is immediate by observing that any node can reach the new merged cluster center say c by first following the path to its old center, say t and then following the path P_t corresponding to the arc in H'' from t to c. By Lemma 3 above, the length of P_t is at most $4L$ and the claim follows.

Lemma 6. *The maximum degree at any node of G induced by the union of paths output by Merge-Centers($\mathcal{C}^*$) is $O(L)$.*

Proof. This is a simple consequence of the performance guarantee of rounding the LP solution obtained for the collection of paths. Since the paths we found pack into the LP solution $2x$ (from the property of the multiflow packing), the expected congestion due to the chosen random paths on any edge e is at most $2x(e)$. From the first constraint in the LP, the expected congestion at any node due to paths incident on it is at most $2L_1 \leq 2L$, by linearity of expectation.

We apply the classic rounding algorithm of [21]. Since the length of each path in the collection is at most $4L$ and the expected congestion is at most $2L$, we obtain that there is a rounding, which can be determined in polynomial time, such that the node congestion (degree) in the rounded solution of at most $6L$.

By Lemma 4, the number of iterations of the main Algorithm 1 is $O(\log k)$ where k is the number of terminals. Lemma 5 guarantees that the subgraph of the final cluster containing all the terminals has distance $O(\log k \cdot L)$ between any pair of terminals. Since the final output is a shortest path tree of this subgraph rooted at r, its diameter is also of the same order. Lemma 6 ensures that the total degree of any node in the subgraph of the final cluster is $O(\log k \cdot L)$, and this is also true for the tree finally output. This completes the proof of Theorem 2. We can derandomize the above randomized algorithm using the standard method of pessimistic estimators [27].

3 Approximating Multicommodity Multicast on Planar Graphs

In this section we prove Theorem 1. Let $G = (V, E)$ be the given planar graph, with $n = |V|$, and let $K = \{(s_i, t_i) : 1 \leq i \leq k\}$ be the set of the k source-destination pairs that need to be connected. Let $\gamma = 1/\log k$. We given a brief overview of our algorithm PlanarMCMulticast, which is fully described in Algorithm 3.

PlanarMCMulticast is a recursive algorithm, breaking the original problem into smaller problems each with at most a constant fraction of the demand pairs in K in each recursive call, thus having $O(\log k)$ depth in the recursion. For a given graph, the algorithm proceeds as follows.

- Find a node separator composed of three shortest paths from an arbitrary vertex [32] to break the graph into pieces each with a constant fraction of the original nodes.
- Solve a generalized Steiner poise LP on the given pairs to identify demand pairs that cross the separator nodes to an extent at least $\Omega(\gamma)$.
- Satisfy these demand pairs by routing their messages from the sources to the separator, moving the messages along the separator (since they are shortest paths, so this movement takes minimal time) and back to the destinations, by scaling the LP values by a factor of $O(\frac{1}{\gamma})$ and using Theorem 2 to find a low poise tree to route to/from the separator.
- For the remaining demand pairs (which are mainly routed within the components after removing the separators), PlanarMCMulticast recurses on the pieces.

Algorithm 3. PlanarMCMulticast(G, K)

```
1: Base case: When $K = \{(s_1, t_1)\}$ has one demand pair, schedule the message on
   the shortest path between the source, $s_1$, and destination, $t_1$.
2: Separate the graph: Define the weight of a node as the number of source-
   destination pairs it is part of, and the weight of a subset of nodes as the sum
   of their weights. Find a 3-path separator $\mathcal{P}$ of $G$, given by shortest paths $P_1$, $P_2$,
   and $P_3$, whose removal partitions the graph into connected components each of
   which has weight at most half that of the graph [32].
3: Partition the terminal pairs: Partition the set $K$ into two subsets, by solving
   the POISE-LP.
   • Let $K_1$ consist of pairs $(s_i, t_i)$ such that in POISE-LP, the fraction of the unit
     flow from $s_i$ to $t_i$ that intersects $\mathcal{P}$ is at least $\gamma$.
   • Let $K_2 = K - K_1$ consist of the remaining pairs, i.e. pairs $(s_i, t_i)$ such that in
     the LP, the fraction of the unit flow from $s_i$ to $t_i$ that intersects $\mathcal{P}$ is less than $\gamma$
4: Scale flow for pairs in $K_1$: For each pair $(s_i, t_i)$ in $K_1$, scale the flow between
   $s_i$ and $t_i$ in the POISE-LP by $\frac{3}{\gamma}$ so there exists a path $P_j$ which intersects a unit
   of this scaled $s_i$-$t_i$ flow; remove other $s_i$-$t_i$ flows that does not intersect $P_j$ up to a
   unit. Assign $(s_i, t_i)$ to a set $S_j$.
5: Create 3 minimum poise Steiner tree problems for $K_1$: For each path $P_j$,
   create a minimum Steiner poise problem as follows: (i) attach, to the graph, an
   auxiliary binary tree $T_j$ with nodes of $P_j$ forming the leaves, and adding new
   dummy internal nodes (This step is similar to [28]); (ii) set the root of the binary
   tree to be the root for the Steiner poise problem, and the terminals to be all the
   $s_i$ and $t_i$ in $S_j$.
6: Round the POISE-LP solution: For each $P_j$, round the LP to obtain a Steiner
   tree $T_j$ of small poise connecting all the terminals in $S_j$ with the root using the
   algorithm from Theorem 2.
7: Construct schedule for $K_1$: Use Lemma 1 on the tree $T_j$ to perform a multi-
   cast between all terminals in it as follows: use the multicast schedule to move the
   messages, from the sources, till they hit the path $P_j$, then move messages along
   the path followed by the multicast schedule in reverse to move them towards the
   destinations. (Moving messages along a path can be achieved by a schedule that
   alternates between the even and odd matchings in the path for as many steps as
   the target length of the schedule)
8: Scale flow for $K_2$: For each pair $(s_i, t_i)$ in $K_2$, remove any flow that intersects $\mathcal{P}$
   and scale the remaining flow (by a factor of at most $\frac{1}{1-\gamma}$) so as to continue to have
   unit total flow between the pair.
9: Recurse for $K_2$: For each connected component $C_j$, let $K_2^j$ denote the subset of
   $K_2$ with both terminals in $C_j$. Run PlanarMCMulticast$(C_j, K_2^j)$ in parallel.
```

The key aspect of planarity that is used here is the structure theorem that planar graphs contain [32] small-size balanced vertex separators that are a combination of three shortest paths starting from any given node.

The analysis of the algorithm are in the full version [19]. Here, we crucially use the fact that the separator paths are shortest paths - for a demand pair (s_i, t_i) let f_i denote the first vertex on the separator path that the message arrives at

Algorithm 4. A gathering procedure for radio gossip in planar graphs.

```
1: Clusters $\mathcal{C}_0 \leftarrow \{V\}$; Vertices $V_0 \leftarrow V$; Graph $G_0 \leftarrow G$; Iteration $i \leftarrow 1$.
2: while $V_{i-1} \neq \emptyset$ do
3:     for all connected component $C \in \mathcal{C}_{i-1}$ do
4:         Choose some $v \in C$. Find shortest paths $p_1, p_2, p_3$ from $v$ that form a 3-path
           separator in $C$ using [32]; Add these to $P_i$, the paths found in the $i$th iteration.
5:         Add $v$ and every $(2L+1)$st vertex along paths $p_1, p_2, p_3$ to $N_i$
6:     end for
7:     Remove the vertices in $P_i$ from $V_{i-1}$ to get $V_i$; Let $G_i$ be $G[V_i]$ and $\mathcal{C}_i$ denote
       the connected components of $G_i$; Increment $i$.
8: end while
9: while $i > 0$ do
10:    Do $2L$ rounds of radio broadcasts in series on nodes that are $2L+1$ apart from
       each other along the paths in $P_i$ to gather all the messages on $P_i$ at the nodes
       $N_i$.
11:    Form $G'_i$ a bipartite graph from $N_i$ to $U_i = \cup_{j=1}^{i-1} P_j$. Add an edge $uv \in E'_i$ if
       there is a path from $u \in N_i$ to $v$ in $G[\mathcal{C}_{i-1} \cup \{v\}]$ of length at most $L$. Find a
       $3L$-matching in $G'$ where every vertex of $U_i$ has degree at most $3L$.
12:    Do up to $L$ rounds of radio broadcast to get the messages from $N_i$ to within
       one node of $U_i$, along the paths in the $3L$-matching found above. Note that the
       messages stay within the component in $\mathcal{C}_{i-1}$ containing $u$ for this part.
13:    Move the messages from the last nodes in $\mathcal{C}_{i-1}$ to their destination nodes in $U_i$
       in the $3L$-matching using at most $9L$ rounds for each of the paths ($27L \log n$
       total).
14:    Decrement $i$
15: end while
```

after leaving source s_i and let l_i denote the last vertex (on the separator path) that the message departs from, on its way to the destination t_i; then f_i and l_i must be at most an additive $O(OPT)$ of the sum of the lengths of the paths from s_i to f_i and l_i to t_i along the separator path, since every demand pair has a path of length $O(OPT)$ between them in the LP solution in this subgraph. Thus in Step 7, we can wait to aggregate all messages from the sources at the separator path, then move all the messages one way along the path and then the opposite way, for as many time steps as the poise of the integral tree, without more than tripling the total schedule.

4 A Polylogarithmic Approximation for Radio Gossip on Planar Graphs

In this section, we present the main ideas for an $O(\log^2 n)$-approximation algorithm for finding a radio gossip schedule on planar graphs. The details of the algorithm are given below and the proof of Theorem 3 is in the full version of the paper [19].

Let $G = (V, E)$ be a given planar graph. Once the messages from all nodes have all been gathered together at a node we can easily broadcast them back out in $O(OPT + \log^2 n)$ rounds using [23]. We focus on gathering the messages together at one node. To do this, we recursively find 3-path separators in the graph [32] to decompose it into connected components. Then, working backwards, we gather messages from the 3-path separators found in an iteration at the nodes of the 3-path separators found in previous iterations, using techniques from telephone multicast [28]. The key properties used in the recursive algorithm are that the number of paths in the separator is a constant 3 and the paths are all shortest paths in the component they separate from some vertex.

References

1. Abraham, I., Gavoille, C.: Object location using path separators. In: PODC, pp. 188–197 (2006)
2. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. J. Comput. Syst. Sci. **43**, 290–298 (1991)
3. Baker, B., Shostak, R.: Gossips and telephones. Discret. Math. **2**(3), 191–193 (1972)
4. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Message multicasting in heterogeneous networks. SIAM J. Comput. **30**(2), 347–358 (2000)
5. Cherkassky, B.: Mnogopolyusnye dvukhproduktovye zadachi [Russian: Multiterminal two commodity problems]. Issledovaniya po Diskretnoi Optimizatsii [Russian: Studies in discrete optimization], pp. 261–289 (1976)
6. Elkin, M., Kortsarz, G.: A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem. SIAM J. Comput. **35**(3), 672–689 (2005)
7. Elkin, M., Kortsarz, G.: Polylogarithmic additive inapproximability of the radio broadcast problem. SIAM J. Discret. Math. **19**(4), 881–899 (2005)
8. Elkin, M., Kortsarz, G.: An approximation algorithm for the directed telephone multicast problem. Algorithmica **45**(4), 569–583 (2006)
9. Elkin, M., Kortsarz, G.: Sublogarithmic approximation for telephone multicast. J. Comput. Syst. Sci. **72**(4), 648–659 (2006)
10. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. Random Struct. Algorithms **1**(4), 447–460 (1990)
11. Fountoulakis, N., Panagiotou, K., Sauerwald, T.: Ultra-fast rumor spreading in social networks. In: SODA 2012, pp. 1642–1660. SIAM (2012)
12. Frank, A.: Connections in Combinatorial Optimization. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, Oxford (2011)
13. Gąsieniec, L.: On efficient gossiping in radio networks. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 2–14. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11476-2_2
14. Gasieniec, L., Peleg, D., Xin, Q.: Faster communication in known topology radio networks. Distrib. Comput. **19**(4), 289–300 (2007)
15. Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: STACS 2011, vol. 9, pp. 57–68 (2011)
16. Grigni, M., Peleg, D.: Tight bounds on minimum broadcast networks. SIAM J. Discret. Math. **4**(2), 207–222 (1991)
17. Hajnal, A., Milner, E.C., Szemerédi, E.: A cure for the telephone disease. Canad. Math. Bull **15**(3), 447–450 (1972)
18. Iglesias, J., Rajaraman, R., Ravi, R., Sundaram, R.: Rumors across radio, wireless, telephone. In: FSTTCS, pp. 517–528 (2015)

19. Iglesias, J., Rajaraman, R., Ravi, R., Sundaram, R.: Plane gossip: Approximating rumor spread in planar graphs. CoRR, abs/1612.01492 (2016)
20. Karp, R., Schindelhauer, C., Shenker, S., Vocking, B.: Randomized rumor spreading. In: FOCS 2000, Washington, DC, USA, pp. 565–574. IEEE (2000)
21. Karp, R.M., Leighton, F.T., Rivest, R.L., Thompson, C.D., Vazirani, U.V., Vazirani, V.V.: Global wire routing in two-dimensional arrays. Algorithmica **2**(1), 113–129 (1987)
22. Kortsarz, G., Peleg, D.: Approximation algorithms for minimum-time broadcast. SIAM J. Discret. Math. **8**(3), 401–427 (1995)
23. Kowalski, D.R., Pelc, A.: Optimal deterministic broadcasting in known topology radio networks. Distrib. Comput. **19**(3), 185–195 (2007)
24. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
25. Lovász, L.: On some connectivity properties of Eulerian graphs. Acta Math. Acad. Sci. Hung. **28**, 129–138 (1976)
26. Manne, F., Wang, S., Xin, Q.: Faster radio broadcast in planar graphs. In: WONS, pp. 9–13 (2008)
27. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge International Series on Parallel Computation. Cambridge University Press, Cambridge (1995)
28. Nikzad, A., Ravi, R.: Sending secrets swiftly: approximation algorithms for generalized multicast problems. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 568–607. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43951-7_48
29. Proskurowski, A.: Minimum broadcast trees. IEEE Trans. Comput. **30**(5), 363–366 (1981)
30. Ravi, R.: Rapid rumor ramification: approximating the minimum broadcast time. In: FOCS, pp. 202–213. IEEE (1994)
31. Ravi, R.: Matching based augmentations for approximating connectivity problems. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 13–24. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_4
32. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM **51**(6), 993–1024 (2004)
33. Tijdeman, R.: On a telephone problem. Nieuw Archief voor Wiskunde **3**(19), 188–192 (1971)

Algorithms and Bounds for Very Strong Rainbow Coloring

L. Sunil Chandran[1], Anita Das[2], Davis Issac[3](✉), and Erik Jan van Leeuwen[4]

[1] Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
sunil@csa.iisc.ernet.in

[2] Infosys Ltd., Bangalore, India
anita_das01@infosys.com

[3] MPI für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
dissac@mpi-inf.mpg.de

[4] Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
e.j.vanleeuwen@uu.nl

Abstract. A well-studied coloring problem is to assign colors to the edges of a graph G so that, for every pair of vertices, all edges of at least one shortest path between them receive different colors. The minimum number of colors necessary in such a coloring is the strong rainbow connection number ($\mathbf{src}(G)$) of the graph. When proving upper bounds on $\mathbf{src}(G)$, it is natural to prove that a coloring exists where, for *every* shortest path between every pair of vertices in the graph, all edges of the path receive different colors. Therefore, we introduce and formally define this more restricted edge coloring number, which we call *very strong rainbow connection number* ($\mathbf{vsrc}(G)$).

In this paper, we give upper bounds on $\mathbf{vsrc}(G)$ for several graph classes, some of which are tight. These immediately imply new upper bounds on $\mathbf{src}(G)$ for these classes, showing that the study of $\mathbf{vsrc}(G)$ enables meaningful progress on bounding $\mathbf{src}(G)$. Then we study the complexity of the problem to compute $\mathbf{vsrc}(G)$, particularly for graphs of bounded treewidth, and show this is an interesting problem in its own right. We prove that $\mathbf{vsrc}(G)$ can be computed in polynomial time on cactus graphs; in contrast, this question is still open for $\mathbf{src}(G)$. We also observe that deciding whether $\mathbf{vsrc}(G) = k$ is fixed-parameter tractable in k and the treewidth of G. Finally, on general graphs, we prove that there is no polynomial-time algorithm to decide whether $\mathbf{vsrc}(G) \leq 3$ nor to approximate $\mathbf{vsrc}(G)$ within a factor $n^{1-\varepsilon}$, unless P = NP.

1 Introduction

The chromatic number is one of the most widely studied properties in graph theory. It has inspired a wealth of combinatorial and algorithmic results, as well as a host of variants. A variant that has recently attracted much interest is

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 625–639, 2018.
https://doi.org/10.1007/978-3-319-77404-6_46

the *rainbow connection number* of a graph, which is an edge coloring property introduced by Chartrand et al. [8] in 2008. Formally, the rainbow connection number $\mathbf{rc}(G)$ of a graph G is the smallest number of colors needed such that there exists a coloring of $E(G)$ with these colors such that, for every pair of vertices, there exists at least one path P between them, such that all edges of P receive different colors. We also say that this path P is *rainbow colored*. The rainbow connection number has attracted much attention, and the exact number is known for a variety of simple graph classes [5,6,8] and the complexity of computing this number was broadly investigated [1,2,4,6,7]. See also the surveys by Li et al. [17–19]. Most recently, in ESA 2016, it was shown that for any $k \geq 2$, deciding whether $\mathbf{rc}(G) \leq k$ (k-Rc) cannot be solved in $2^{o(n^{3/2})}$ or $2^{o(m/\log m)}$ time, where $n = |V(G)|$ and $m = |E(G)|$, unless ETH fails [15].

To prove an upper bound on $\mathbf{rc}(G)$, the choice of the path P that is rainbow colored is crucial. The analysis would seem simpler when we are able to choose P as a shortest path between its two endpoints. This leads to the definition of the *strong rainbow connection number* of a graph. Formally, the strong rainbow connection number $\mathbf{src}(G)$ of a graph G is the smallest number of colors needed such that there exists a coloring of $E(G)$ with these colors such that, for every pair of vertices, there exists at least one *shortest* path P between them, such that all edges of P receive different colors. Clearly, $\mathbf{src}(G) \geq \mathbf{rc}(G)$, and both parameters are at least the diameter of G. Moreover, $\mathbf{rc}(G) = 2$ if and only if $\mathbf{src}(G) = 2$ [4]. Nontrivial upper bounds on $\mathbf{src}(G)$ are known for several simple graph classes such as cycles, wheels, and complete bipartite graphs [8] and block graphs [16]. It is also known that deciding whether $\mathbf{src}(G) \leq k$ (k-Src) is NP-hard even for $k = 2$ [4]. The problem of deciding whether $\mathbf{src}(G) \leq k$ remains NP-complete even for bipartite graphs and split graphs [1,14]. In fact, $\mathbf{src}(G)$ cannot be approximated in polynomial time within a factor $n^{1/2-\varepsilon}$ for any $\varepsilon > 0$, unless P = NP, even for split and bipartite graphs [1,14].[1]

The lack of combinatorial bounds on $\mathbf{src}(G)$ for specific graph classes G (the recent survey by Li and Sun [19] cites only three papers) is somewhat surprising compared to the vast literature for $\mathbf{rc}(G)$ (see the surveys [17–19]). Li and Sun [19] explain this by the fact that $\mathbf{src}(G)$ is not a monotone graph property, and thus investigating $\mathbf{src}(G)$ is much harder than investigating $\mathbf{rc}(G)$. Hence, it is a major open question to prove upper bounds on $\mathbf{src}(G)$.

In this paper, we make significant progress on this question. We observe that to prove upper bounds on $\mathbf{src}(G)$, it suffices to prove the existence of a coloring where all edges of not just one, but of *all* shortest paths between two vertices receive different colors. Therefore, we define the *very strong rainbow connection number* $\mathbf{vsrc}(G)$ of a graph G, which is the smallest number of colors for which there exists a coloring of $E(G)$ such that, for every pair of vertices and *every shortest* path P between them, all edges of P receive different colors. We call a coloring that achieves this property a *very strong rainbow coloring* of the graph. We also call the problem of deciding whether $\mathbf{vsrc}(G) \leq k$ the k-Vsrc problem.

[1] [1,14] mention NP $\neq$ ZPP as the complexity assumption but one can use P $\neq$ NP because of [23].

Our Results. We prove the first combinatorial upper bounds on $\mathbf{vsrc}(G)$ for several graph classes. These immediately imply upper bounds on $\mathbf{src}(G)$ for the same graph classes. In particular, we show upper bounds that are linear in $|V(G)|$ (improving from the trivial bound of $|E(G)|$) if G is a chordal graph, a circular arc graph, or a disk graph. We also make progress on the following conjecture:

Conjecture 1.1 ([16]). For any connected graph G, $\mathbf{src}(G) \leq |V(G)| - \chi(G) + 1$ where $\chi(G)$ denotes the chromatic number of G.

We show that the conjecture holds for the class of chordal graphs in Corollary 2.4.

Conversely, we prove that a bound on $\mathbf{vsrc}(G)$ implies that G should be highly structured: the neighborhood of every vertex can be partitioned into $\mathbf{vsrc}(G)$ cliques. For further details, we refer to Sect. 2.

In the second part of the paper, we address the computational complexity of k-VSRC. To start our investigation, we prove hardness results on general graphs.

Theorem 1.2. *3-VSRC is NP-complete. Moreover, there is no polynomial-time algorithm that approximates* $\mathbf{vsrc}(G)$ *within a factor* $|V(G)|^{1-\varepsilon}$ *for any* $\varepsilon > 0$*, unless P = NP.*

This result implies that k-VSRC is not fixed-parameter tractable when parameterized by k, unless P = NP. In order to prove the theorem, we show a nontrivial connection to the clique partition number of a graph.

We remark that, in contrast to the NP-complete 2-RC and 2-SRC problems, 2-VSRC can be solved in polynomial time (see Sect. 5 for the proof). Together with Theorem 1.2, this gives a dichotomy result for the complexity of k-VSRC.

Proposition 1.3. *Let G be any graph. Then* 2-VSRC *can be decided in polynomial time.*

We then study the complexity of determining $\mathbf{vsrc}(G)$ for graphs of bounded treewidth. This is a major open question also for $\mathbf{src}(G)$ and $\mathbf{rc}(G)$ [16], which are only known to be solvable in polynomial time on graphs of treewidth 1. We mention that no results for graphs of higher treewidth are known, even for outerplanar or cactus graphs. However, for the slightly different problem of deciding whether an already given coloring forms a (strong) rainbow coloring of a given graph, a polynomial-time algorithm for cactus graphs and an NP-hardness result for outerplanar graphs are known [22]. With this in mind, we focus on cactus graphs and make the first progress towards understanding the complexity of rainbow coloring problems, in particular of computing $\mathbf{vsrc}(G)$, on graphs of treewidth 2 with the following result.

Theorem 1.4. *Let G be any cactus graph. Then* $\mathbf{vsrc}(G)$ *can be computed in polynomial time.*

Our algorithm relies on an extensive characterization result for the behavior of very strong rainbow colorings on cactus graphs. Since a cactus graph consists of bridges, even cycles, and odd cycles, we analyze the behavior of any very strong rainbow coloring of the graph with respect to these structures. We show that color repetition can mostly occur only within an odd cycle or even cycle. Odd cycles can repeat some colors from outside but we characterize how they can be repeated. However, our arguments are not sufficient to derive a completely combinatorial bound. Instead, we must find a maximum matching in a well-chosen auxiliary graph to compute the very strong rainbow connection number.

We also observe that $\mathbf{vsrc}(G)$ can be computed efficiently for graphs having bounded treewidth, when $\mathbf{vsrc}(G)$ itself is small. In contrast to known results for the (strong) rainbow connection number [9], we present an algorithm that does not rely on Courcelle's theorem. (See Sect. 5 for details.)

Theorem 1.5. *k-VSRC is fixed-parameter tractable when parameterized by $k+t$, where $t-1$ is the treewidth of the input graph.*

Preliminaries. We consider simple, undirected graphs and use standard notation for graphs. Given a universe $\mathcal{U} = \{x_1, x_2, \dots, x_n\}$ and a family $\mathcal{F} = \{S_1, S_2, \dots, S_t\}$ of subsets of $\mathcal{U}$, the *intersection graph* $G(\mathcal{F})$ of $\mathcal{F}$ has vertex set $\{v_1, \dots, v_t\}$, and there is an edge between two vertices v_i, v_j if and only if $S_i \cap S_j \neq \emptyset$. We call $\mathcal{F}$ a *representation* of $G(\mathcal{F})$. An *interval graph* is an intersection graph of intervals on the real line. The interval graph is *proper* if it has a representation by intervals where no interval is properly contained in another. A *circular arc graph* is an intersection graph of arcs of a circle. A chordal graph is an intersection graph of subtrees of a tree. A *block* of a graph is a maximal 2-connected component. In a *cactus graph*, each block of the graph is a cycle or an edge; equivalently, every edge belongs to at most one cycle.

For a graph G, let $\hat{G}$ denote the graph obtained by adding a new vertex $\hat{u}$ to G such that $\hat{u}$ is adjacent to all vertices of G, i.e., $\hat{u}$ is a universal vertex in $\hat{G}$.

Finally, we use $\omega(G)$ to denote the maximum size of any clique in graph G. We use $d(u, v)$ to denote the length of a shortest path between vertices u and v.

2 Combinatorial Results

We show several upper and lower bounds on $\mathbf{vsrc}(G)$, both for general graphs and for graphs G that belong to a specific graph class. Crucial in our analysis are connections between very strong rainbow colorings and decompositions of the input graph into cliques. We use $\mathbf{cp}(G)$ to denote the *clique partition number* (or clique cover number) of G, the smallest number of subsets of $V(G)$ that each induce a clique in G and whose union is $V(G)$. $\hat{G}$ used in the following lemma (defined in the preliminaries) is important for our hardness reductions.

Lemma 2.1. *Let G be any graph. Then*

1. $\mathbf{src}(G) \leq \mathbf{vsrc}(G) \leq \mathbf{cp}(G)(\mathbf{cp}(G)+1)/2$.
2. $\mathbf{src}(\hat{G}) \leq \mathbf{vsrc}(\hat{G}) \leq \mathbf{cp}(G)(\mathbf{cp}(G)+1)/2$.

Proof. Let $\mathcal{C} = C_1, \dots, C_r$ be the set of cliques in an optimal clique partition of G; that is, $r = \mathbf{cp}(G)$. For a vertex v, let $c(v)$ denote the clique in $\mathcal{C}$ that contains v. We define the set of colors as $\mathcal{P}_{\leq 2}(\mathcal{C}) \setminus \{\emptyset\}$, the set of subsets of $\mathcal{C}$ of size 1 or 2. We then color any edge $uv \in E(G)$ by $\{c(u), c(v)\}$. For sake of contradiction, suppose that this does not constitute a very strong rainbow coloring of G. Then there exist two vertices $s, t \in V(G)$, a shortest path P between s and t, and two edges $uv, wx \in E(P)$ that received the same color. If $c(u) = c(v)$, then $c(w) = c(x)$, meaning that P uses two edges of the same clique. Then P can be shortcut, contradicting that P is a shortest path between s and t. Hence, $c(u) \neq c(v)$ and thus $c(w) \neq c(x)$. Without loss of generality, $c(u) = c(w)$ and thus $c(v) = c(x)$. Then either the edge uw or the edge vx will shortcut P, a contradiction. Hence, $\mathbf{vsrc}(G) \leq \mathbf{cp}(G)(\mathbf{cp}(G) + 1)/2$ by the set of colors used. To see the second part of the lemma, color edges $\hat{u}v$ incident on the universal vertex $\hat{u}$ in $\hat{G}$ by $c(v)$ in addition to the above coloring. Suppose this was not a very strong rainbow coloring of $\hat{G}$. Then there exists vertices u, v such that $u\hat{u}v$ is a shortest path and $u\hat{u}$ and $v\hat{u}$ are colored the same. But then u and v are in the same clique C_i in $\mathcal{C}$. But then uv can shortcut $u\hat{u}v$, a contradiction. □

The following lemma is more consequential for our upper bounds. We use $\mathbf{is}(G)$ to denote the smallest size of the universe in any intersection graph representation of G, and $\mathbf{ecc}(G)$ to denote the smallest number of cliques needed to cover all edges of G. It is known that $\mathbf{is}(G) = \mathbf{ecc}(G)$ [20].

Lemma 2.2. *Let G be any graph. Then $\mathbf{vsrc}(G) \leq \mathbf{is}(G) = \mathbf{ecc}(G)$.*

Proof. Let $\mathcal{U} = \{x_1, x_2, \dots, x_n\}$ be a universe and let $\mathcal{F} = \{S_1, S_2, \dots, S_m\}$ be a family of subsets of $\mathcal{U}$, such that G is the intersection graph of $\mathcal{F}$ and $|\mathcal{U}| = \mathbf{is}(G)$. Let v_i be the vertex of G corresponding to the set S_i. We consider $x_1, x_2, \dots, x_n$ as colors, and color an edge between vertices v_i and v_j with any $x \in S_i \cap S_j$ (note that this intersection is nonempty by the presence of the edge). Suppose for sake of contradiction that this is not a very strong rainbow coloring of G. Then there exist two vertices $s, t \in V(G)$, a shortest path P between s and t, and two edges $v_i v_j$ and $v_a v_b$ in P that received the same color x. By the construction of the coloring, this implies that $x \in S_i \cap S_j \cap S_a \cap S_b$. Hence, v_i, v_j, v_a, v_b induce a clique in G. But then the path P can be shortcut, a contradiction. □

A similar lemma for $\mathbf{src}(G)$ was proved independently by Lauri [16, Proposition 5.3].

Corollary 2.3. *Let G be any graph. Then $\mathbf{vsrc}(G) \leq \min\{\lfloor |V(G)|^2/4 \rfloor, |E(G)|\}$.*

Proof. Directly from $\mathbf{ecc}(G) \leq \min\{\lfloor |V(G)|^2/4 \rfloor, |E(G)|\}$ for any graph [10]. □

Corollary 2.4. *Let G be any graph.*

1. *If G is chordal, then $\mathbf{src}(G) \leq \mathbf{vsrc}(G) \leq |V(G)| - \omega(G) + 1$.*
2. *If G is circular-arc, then $\mathbf{src}(G) \leq \mathbf{vsrc}(G) \leq |V(G)|$.*
3. *$\mathbf{src}(L(G)) \leq \mathbf{vsrc}(L(G)) \leq |V(G)|$, where $L(G)$ is the line graph of G.*

These bounds are (almost) tight in general.

Proof. In each of the three cases, we express the graph as an intersection graph over a suitable universe, and then by Lemma 2.2, we get that the size of the universe is an upper bound on **vsrc** of the graph.

Every chordal graph is the intersection graph of subtrees of a tree [12]. It is also known that the number of vertices of this tree only needs to be at most $|V(G)| - \omega(G) + 1$. (For completeness, we provide a proof of this in the full version.)

For a circular arc graph G, consider any set of arcs whose intersection graph is G. We now construct a different intersection representation. Take the set of second (considering a clockwise ordering of points) endpoints of all arcs as the universe $\mathcal{U}$. Take $S_i \subseteq \mathcal{U}$ as the set of clockwise endpoints contained in the i-th arc. It is easy to see that G is the intersection graph of $\mathcal{F} = \{S_1, S_2, \ldots, S_n\}$.

Finally, consider $L(G)$. We construct an intersection representation with universe $V(G)$. For each $uv \in E(G)$, let $S_{uv} = \{u, v\}$. Then $L(G)$ is the intersection graph of $\mathcal{F} = \{S_e : e \in E(G)\}$.

The (almost) tightness follows from $\mathbf{vsrc}(G) = |V(G)| - 1$ and $\mathbf{vsrc}(L(G)) = |V(G)| - 2$ for any path G. Paths are both chordal and circular-arc. □

In the remainder, we consider a natural generalization of line graphs. A graph is *k-perfectly groupable* if the neighborhood of each vertex can be partitioned into k or fewer cliques. It is well known that line graphs are 2-perfectly groupable. A graph is *k-perfectly orientable* if there exists an orientation of its edges such that the outgoing neighbors of each vertex can be partitioned into k or fewer cliques. Clearly, any k-perfectly groupable graph is also k-perfectly orientable. Many geometric intersection graphs, such as disk graphs, are known to be k-perfectly orientable for small k [13].

Corollary 2.5. *Let G be any k-perfectly orientable graph. Then,* $\mathbf{src}(G) \leq \mathbf{vsrc}(G) \leq k|V(G)|$.

Proof. Consider any orientation of the edges of G such that the outgoing neighbors of each vertex can be partitioned into k or fewer cliques. For a given vertex v, let $C(v)$ denote the set of cliques induced by its outgoing neighbors, where v is added to each of those cliques. Observe that $\bigcup_{v \in V(G)} C(v)$ is an edge clique cover of G, because every edge is outgoing from some vertex v and will thus be covered by a clique in $C(v)$. Hence, $\mathbf{vsrc}(G) \leq \mathbf{ecc}(G) \leq k|V(G)|$. □

Since any k-perfectly groupable graph is also k-perfectly orientable, the above bound also applies to k-perfectly groupable graphs. In this context, we prove an interesting converse of the above bound.

Lemma 2.6. *Let G be any graph. If* $\mathbf{vsrc}(G) \leq k$, *then G is k-perfectly groupable.*

Proof. Consider an optimal very strong rainbow coloring μ of G. Consider an arbitrary vertex v of G and let c be any color used in μ. Define the set $Q(c) = \{u \in N(v) : \mu(vu) = c\}$. Suppose there exist two non-adjacent vertices u, w in $Q(c)$. Then uvw is a shortest path between u and w, and thus uv and vw cannot have the same color, a contradiction. Hence, for each color c used in μ, $Q(c)$ is a clique. Since the number of colors is at most k, the edges incident on v can be covered with at most k cliques. Hence, G is k-perfectly groupable. □

3 Hardness Results

The hardness results lean heavily on the combinatorial bounds of the previous section. In this section, we use $\hat{G}$ (see the preliminaries for the definition) extensively. We need the following bound, which strengthens Lemma 2.1.

Lemma 3.1. *Let G be any graph. If* $\mathbf{cp}(G) \leq 3$*, then* $\mathbf{vsrc}(\hat{G}) \leq 3$*.*

Proof. Let C_1, C_2, and C_3 be three cliques into which $V(G)$ is partitioned. We will color $\hat{G}$ with three colors, say c_1, c_2, and c_3, as follows. For each edge with both endpoints in C_i for $1 \leq i \leq 3$, color it with c_i. For each edge vw with $v \in C_i$, $w \in C_j$ such that $1 \leq i < j \leq 3$, color it with c_k, where $k \in \{1, 2, 3\} \setminus \{i, j\}$. Finally, for each edge $\hat{u}v$ with $v \in C_i$ for $1 \leq i \leq 3$, color it with c_i.

Suppose this is not a very strong rainbow coloring of $\hat{G}$. Since the diameter of $\hat{G}$ is at most 2, there exists a shortest path xyz with xy and yz having the same color. However, if xy and yz have the same color, at least two of x, y and z are in the same C_i for $1 \leq i \leq 3$ and the third one is either $\hat{u}$ or in C_i itself. Then, we can shortcut xyz by xz, a contradiction. Hence $\mathbf{vsrc}(\hat{G}) \leq 3$. □

Proof (of Theorem 1.2). We first prove that 3-Vsrc is NP-complete. We reduce from the NP-hard 3-Coloring problem [11]. Let G be an instance of 3-Coloring. Let H be the complement of G. We claim that $\mathbf{vsrc}(\hat{H}) = 3$ if and only if G is 3-colorable. Indeed, if $\mathbf{vsrc}(\hat{H}) \leq 3$, then $\hat{H}$ is 3-perfectly groupable by Lemma 2.6. In particular, the neighborhood of $\hat{u}$ (the universal vertex in $\hat{H}$) can be partitioned into at most 3 cliques. These cliques induce disjoint independent sets in G that cover $V(G)$, and thus G is 3-colorable. For the other direction, note that if G is 3-colorable, then $\mathbf{cp}(H) \leq 3$, and by Lemma 3.1, $\mathbf{vsrc}(\hat{H}) \leq 3$.

To prove the hardness of approximation, we recall that there exists a polynomial-time algorithm that takes a Sat formula ψ as input and produces a graph G as output such that if ψ is not satisfiable, then $\mathbf{cp}(G) \geq |V(G)|^{1-\varepsilon}$, and if ψ is satisfiable, then $\mathbf{cp}(G) \leq |V(G)|^{\varepsilon}$ [23, Proof of Theorem 2]. Consider the graph $\hat{G}$ and let n denote the number of its vertices. Then

$$\psi \text{ not satisfiable} \Rightarrow \mathbf{cp}(G) \geq (n-1)^{1-\varepsilon} \Rightarrow \mathbf{vsrc}(\hat{G}) \geq (n-1)^{1-\varepsilon}$$

$$\psi \text{ satisfiable} \Rightarrow \mathbf{cp}(G) \leq (n-1)^{\varepsilon} \Rightarrow \mathbf{vsrc}(\hat{G}) \leq (n-1)^{2\varepsilon}$$

because Lemma 2.6 implies that $\mathbf{vsrc}(\hat{G}) \geq \mathbf{cp}(G)$, and by Lemma 2.1. The result follows by rescaling ε. □

4 Algorithm for Cactus Graphs

Let G be the input cactus graph. We first prove several structural properties of cactus graphs, before presenting the actual algorithm.

4.1 Definitions and Structural Properties of Cactus Graphs

We make several structural observations related to cycles. For a vertex v and a cycle C containing v, we define $\mathsf{S}(v, C)$ as the vertices of G that are reachable from v without using any edge of C.

Observation 4.1. *For any cycle C in G, $\{\mathsf{S}(v,C) : v \in V(C)\}$ is a partition of $V(G)$.*

From Observation 4.1, we have that for any fixed $u \in V(G)$ and any fixed cycle C of G, there exists a unique vertex $v \in V(C)$ such that $u \in \mathsf{S}(v,C)$. We denote that unique vertex v by $\mathsf{g}(u,C)$.

Observation 4.2. *Let $u \in V(G)$ and let C be a cycle in G. Let $w \in V(C)$ and let $x_1x_2\dots x_r$ be a path from u to w where $x_1 = u$ and $x_r = w$. Let i^* be the smallest i such that $x_i \in V(C)$. Then, $x_{i^*} = \mathsf{g}(u,C)$. In simpler words, any path from u to any vertex in C enters C through $\mathsf{g}(u,C)$.*

Observation 4.3. *For any cycle C in G and for any $uv \in E(G) \setminus E(C)$, $\mathsf{g}(u,C) = \mathsf{g}(v,C)$.*

We now consider even cycles. For an edge uv in an even cycle C, we define its *opposite edge*, denoted by $\mathsf{eopp}(uv)$, as the unique edge $xy \in E(C)$ such that $d(u,x) = d(v,y)$. Note that $\mathsf{eopp}(\mathsf{eopp}(e)) = e$. Call the pair of edges e and $\mathsf{eopp}(e)$ an *opposite pair*. Each even cycle C has exactly $\frac{|C|}{2}$ opposite pairs.

Lemma 4.4. *Let C be an even cycle. For any vertex $x \in V(G)$ and edge $uv \in E(C)$, either there is a shortest path between x and u that contains uv or there is a shortest path between x and v that contains uv.*

Proof. Let $w = \mathsf{g}(x,C)$. Then, w cannot be equidistant from u and v, because otherwise C is an odd cycle. Suppose that $d(w,u) < d(w,v)$. Then a shortest path from w to u appended with the edge uv gives a shortest path between w and v. Now, due to Observation 4.2, if we append a shortest path between x and w with a shortest path between w and v, we get a shortest path between x and v. Thus there is a shortest path between x and v that contains uv. If $d(w,u) > d(w,v)$, then we get the other conclusion of the lemma. □

We then consider odd cycles in more detail. For any edge e in an odd cycle C, there is a unique vertex in C, which is equidistant from both endpoints of e. We call this vertex the *opposite vertex* of e and denote it as $\mathsf{vopp}(e)$. We call $\mathsf{OS}(e) = G[\mathsf{S}(\mathsf{vopp}(e), C)]$ the *opposite subgraph* of e. See Fig. 1.

Lemma 4.5. *Let C be an odd cycle and $uv \in E(C)$. For any vertex $x \in V(G) \setminus V(\mathsf{OS}(uv))$, either there is a shortest path between x and u that contains uv or there is a shortest path between x and v that contains uv.*

Proof. Let $w = \mathsf{g}(x, C)$. Since $x \notin V(\mathsf{OS}(uv))$, $w \neq \mathsf{vopp}(uv)$. Hence, w cannot be equidistant from u and v. So, the same arguments as in Lemma 4.4 complete the proof. □

Lemma 4.6. *Let e be any edge in an odd cycle of G for which $\mathsf{vopp}(e)$ has degree more than 2. Then $\mathsf{OS}(e)$ contains a bridge, or an even cycle, or an edge e' in an odd cycle for which $\mathsf{vopp}(e')$ has degree 2.*

Proof. Suppose this is not the case. We define a sequence $e_1, e_2, \ldots$ of edges by the following procedure. Let $e_1 = e$. Given e_i, we define e_{i+1} as follows. By assumption and the definition of cactus graphs, e_i is contained in an odd cycle, which we denote by C_i, and $\mathsf{vopp}(e_i)$ has degree more than 2. Choose e_{i+1} as any edge incident on $\mathsf{vopp}(e_i)$ that is not in C_i. However, observe that $\mathsf{OS}(e_{i+1}) \subset \mathsf{OS}(e_i)$ by the choice of e_{i+1}. Hence, this is an infinite sequence, which contradicts the finiteness of $E(G)$. □

4.2 Properties of Very Strong Rainbow Colorings of Cactus Graphs

We initially partition the edges of G into three sets: E_{bridge}, E_{even}, and E_{odd}. The set E_{bridge} consists of those edges that are not in any cycle. In other words, E_{bridge} is the set of bridges in G. By definition, each of the remaining edges is part of exactly one cycle. We define E_{even} as the set of all edges that belong to an even cycle, and E_{odd} as the set of all edges that belong to an odd cycle. Note that E_{bridge}, E_{even}, and E_{odd} indeed induce a partition of $E(G)$. We then partition E_{odd} into two sets: E_{opp} and E_{rem}. An edge $e \in E_{\mathsf{odd}}$ is in E_{opp} if $\mathsf{vopp}(e)$ is not a degree-2 vertex and in E_{rem} otherwise. See Fig. 1. We analyze each of these sets in turn, and argue how an optimal VSRC might color them.

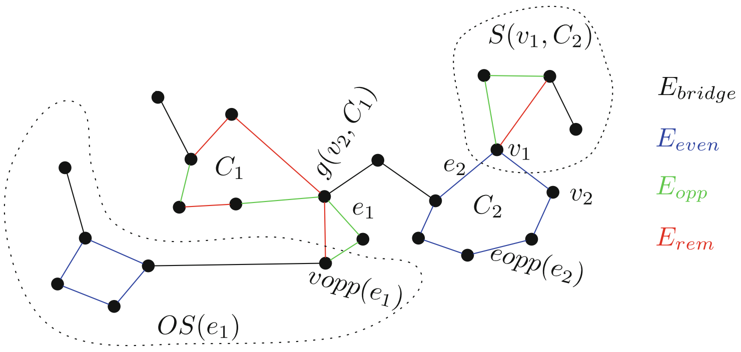


Fig. 1. An example of a cactus graph and related definitions.

Two edges e_1 and e_2 are called *conflicting* if there is a shortest path in the graph which contains both e_1 and e_2. Two conflicting edges must have different colors in any VSRC. We now exhibit several classes of conflicting pairs of edges.

Lemma 4.7. [2]*Any VSRC of G colors the edges of E_{bridge} with distinct colors.*

Proof. Consider $uv, xy \in E_{\mathsf{bridge}}$. We prove that uv and xy are conflicting, i.e. there is a shortest path in G which contains both uv and xy. Since uv is a bridge, we can assume without loss of generality that any path between u and y uses the edge uv. Similarly, since xy is a bridge, we can assume without loss of generality that any path between y and u uses the edge xy. Hence, the shortest path from u to y uses both uv and xy. Hence, uv and xy are conflicting. ☐

Lemma 4.8. *Let $e_1 \in E_{\mathsf{bridge}}$ and $e_2 \in E_{\mathsf{even}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let C be the cycle containing e_2. Let $e_1 = xy$ and $e_2 = uv$. Since xy is a bridge, we can assume w.l.o.g. that any path from x to any vertex in C contains xy. Due to Lemma 4.4, we can assume w.l.o.g. that there is a shortest path from x to v that contains uv. Thus we have a shortest path which contains both uv and xy, which means that uv and xy are conflicting. ☐

Observation 4.9. *Let e_1 and e_2 be edges in an even cycle C of G such that $e_1 \neq \mathsf{eopp}(e_2)$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Lemma 4.10. *Let e_1 and e_2 be edges in two different even cycles C_1 and C_2 of G. Then any VSRC of G colors uv and xy with different colors.*

Proof. Let $e_1 = uv$ and $e_2 = xy$. Let $z = \mathsf{g}(u, C_2)$ and $w = \mathsf{g}(x, C_1)$. By Observation 4.3, $\mathsf{g}(v, C_2) = z$ and $\mathsf{g}(y, C_1) = w$. Due to Lemma 4.4, we can assume w.l.o.g. that there is a shortest path P_1 between z and x containing xy and that there is a shortest path P_2 between w and u containing uv. Let P_3 be a shortest path between w and z. Then $P_1 \cup P_3 \cup P_2$ gives a shortest path between u and x that contains both uv and xy. Hence, e_1 and e_2 are conflicting. ☐

Lemma 4.11. *Let $e_1 \in E_{\mathsf{bridge}} \cup E_{\mathsf{even}}$ and $e_2 \in E_{\mathsf{rem}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let $e_1 = xy$ and $e_2 = uv$, let C be the odd cycle containing e_2, and let $w = \mathsf{g}(x, C)$. By Observation 4.3, $w = \mathsf{g}(y, C)$. In other words, $x, y \in \mathsf{S}(w, C)$. Note that w is not a degree-2 vertex, because there are at least two vertices in $\mathsf{S}(w, C)$. Hence, $w \neq \mathsf{vopp}(uv)$ by the definition of E_{rem}. Hence, by Lemma 4.5, w.l.o.g. there is a shortest path P_1 from w to u that contains uv.

We now consider two cases, depending on whether $e_1 \in E_{\mathsf{bridge}}$ or $e_1 \in E_{\mathsf{even}}$. First, suppose that $e_1 \in E_{\mathsf{bridge}}$. Since xy is a bridge, we can assume w.l.o.g. that any shortest path from x to w contains xy. Let P_2 be such a shortest path. By Observation 4.2, if we append a shortest path from x to w with a shortest path

[2] This lemma holds for any graph, not necessarily cactus.

from w to u, we get a shortest path from x to u. Thus, $P_1 \cup P_2$ is a shortest path from x to u containing xy and uv. Hence, e_1 and e_2 are conflicting.

Suppose that $e_1 \in E_{\mathsf{even}}$. Let C' be the even cycle containing e_1. Let $z = \mathsf{g}(v, C')$. From Lemma 4.4, we can assume w.l.o.g. that there is a shortest path from z to x that contains xy. Let this shortest path be P_3. Let P_4 be a shortest path between w and z. By Observation 4.2, $P_3 \cup P_4 \cup P_1$ is a shortest path between x and u that contains xy and uv. Hence, e_1 and e_2 are conflicting. □

Lemma 4.12. *Let C_1 and C_2 be two distinct odd cycles and let $e_1 \in E(C_1) \cap E_{\mathsf{rem}}$ and $e_2 \in E(C_2) \cap E_{\mathsf{rem}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let $e_1 = xy$ and $e_2 = uv$, and let $w = \mathsf{g}(x, C_2)$. By Observation 4.3, $w = \mathsf{g}(y, C_2)$. Let $z = \mathsf{g}(u, C_1)$. By Observation 4.3, $z = \mathsf{g}(v, C_1)$. That is, $x, y \in \mathsf{S}(w, C_2)$ and $u, v \in \mathsf{S}(z, C_1)$. Note that w and z are not degree-2 vertices, because there are at least two vertices in $\mathsf{S}(w, C_2)$ and $\mathsf{S}(z, C_1)$. Hence, $w \neq \mathsf{vopp}(uv)$ and $z \neq \mathsf{vopp}(xy)$ by the definition of E_{rem}. Hence, by Lemma 4.5, we can assume w.l.o.g. that there is a shortest path P_1 from u to w that contains uv and there is a shortest path P_2 from z to x that contains xy. Let P_3 be a shortest path from w to z. By Observation 4.2, $P_1 \cup P_2 \cup P_3$ is a shortest path from x to u containing xy and uv. Hence, e_1 and e_2 are conflicting. □

Finally, we prove the existence of some non-conflicting pairs of edges.

Lemma 4.13. *For any $e_1 \in E_{\mathsf{opp}}$ and $e_2 \in \mathsf{OS}(e_1)$, e_1 and e_2 are not conflicting.*

Proof. Let $e_1 = uv$, $e_2 = xy$, and let C be the odd cycle containing e_1. For sake of contradiction, suppose that uv and xy are conflicting. Assume w.l.o.g. that there is a shortest path P from x to v which contains uv and xy. From Observation 4.2, P contains a subpath P' from $\mathsf{g}(x, C)$ to v. Clearly, P' contains uv. Also, $\mathsf{g}(x, C) = \mathsf{vopp}(uv)$, because $x \in \mathsf{OS}(uv)$. However, recall that $\mathsf{vopp}(uv)$ is equidistant from u and v. Hence, any shortest path from $\mathsf{vopp}(uv)$ to v does not contain uv, which contradicts the existence of P'. □

4.3 Algorithm

Based on the results of the previous two subsections, we now describe the algorithm for cactus graphs. First, we color the edges of E_{bridge} with unique colors. By Lemma 4.7, no VSRC can use less colors to color E_{bridge}.

Next, we color the edges in E_{even} using colors that are distinct from those we used before. This will not harm the optimality of the constructed coloring, because of Lemma 4.8. Moreover, we use different colors for different even cycles, which does not harm optimality by Lemma 4.10. We then introduce a set of $\frac{|C|}{2}$ new colors for each even cycle C. For an opposite pair, we use the same color, and we color each opposite pair with a different color. Thus we use $\frac{|C|}{2}$ colors for each even cycle C. By Observation 4.9, no VSRC can use less colors to color C.

Next, we will color the edges in E_{rem} using colors that are distinct from those we used before. This will not harm the optimality of the constructed coloring, because of Lemma 4.11. For each odd cycle, we use a different set of colors. This will not harm the optimality of the constructed coloring, because of Lemma 4.12.

For each odd cycle C, we construct an auxiliary graph H_C for $E_{\mathsf{rem}} \cap C$ as follows. Let $V(H_C) = E_{\mathsf{rem}} \cap C$ and let $E(H_C) = \{e_1e_2 : e_1, e_2 \in V(H_C); e_1$ and e_2 are not conflicting in $G\}$.

Lemma 4.14. $\Delta(H_C) \leq 2$.

Proof. It is easy to observe that in any odd cycle C, for any $e \in E(C)$, there are only two other edges in C that are not conflicting with e. □

Let M_C be a maximum matching of H_C. We can compute M_C in linear time, since $\Delta(H_C) \leq 2$. For an $e_1e_2 \in M_C$, color e_1 and e_2 with the same, new color. Then color each $e \in E_{\mathsf{rem}} \cap C$ that is unmatched in M_C, each using a new color.

Lemma 4.15. *The procedure for coloring $E_{\mathsf{rem}} \cap C$ gives a coloring of the edges in $E_{\mathsf{rem}} \cap C$ such that no conflicting edges are colored the same. Moreover, no VSRC of G can use less colors to color $E_{\mathsf{rem}} \cap C$ than used by the above procedure.*

Proof. Suppose two conflicting edges $e_1, e_2 \in E_{\mathsf{rem}} \cap C$ were colored the same. Then the corresponding vertices e_1 and e_2 were matched to each other in M_C. Hence, e_1 and e_2 are adjacent in H_C, meaning that e_1 and e_2 did not conflict each other in G, which is a contradiction. Hence, we have proved that no conflicting edges were given the same color by the procedure.

Now, consider any VSRC μ of G which colored $E_{\mathsf{rem}} \cap C$ with fewer colors than by our procedure. Observe that for any edge e in an odd cycle, there are only two other edges (say e_a and e_b) that are not conflicting with e. Moreover, e_a and e_b are conflicting with each other. This means that μ can use each color for at most two edges of $E_{\mathsf{rem}} \cap C$. Suppose there are k_1 colors that are assigned to two edges in $E_{\mathsf{rem}} \cap C$ by μ. Each pair of edges colored the same should be non-conflicting and hence have an edge between them in H_C. So, taking all pairs colored the same induces a matching of size k_1 of H_C. Then $k_1 \leq |M_C|$, because M_C is a maximum matching of H_C. But then the number of colors used by μ is equal to $k_1 + (|E_{\mathsf{rem}} \cap C| - 2k_1) = |E_{\mathsf{rem}} \cap C| - k_1$. The number of colors used by our procedure is $|M_C| + |E_{\mathsf{rem}} \cap C| - 2|M_C| = |E_{\mathsf{rem}} \cap C| - |M_C| \leq |E_{\mathsf{rem}} \cap C| - k_1$. Hence, we use at most the number of colors used by μ. □

Finally, we color the edges of E_{opp} without introducing new colors. Indeed, for every $e \in E_{\mathsf{opp}}$, it follows from Lemma 4.6 that there exists an edge $e' \in E(\mathsf{OS}(e)) \cap (E_{\mathsf{bridge}} \cup E_{\mathsf{even}} \cup E_{\mathsf{rem}})$, which does not conflict with e by Lemma 4.13. Since e' is already colored, say by color c, then we can simply re-use that color c for e. Indeed, suppose for sake of contradiction that there is a shortest path P between two vertices x, y that contains e and that contains another edge e'' using the color c. By Lemma 4.13, $e'' \notin \mathsf{OS}(e)$. This implies that $e'' \notin E_{\mathsf{bridge}} \cup E_{\mathsf{even}} \cup E_{\mathsf{rem}}$ by the choice of c and the construction of the coloring. Hence, $e'' \in E_{\mathsf{opp}}$. However, by a similar argument, e'' can only receive color c if $e' \in \mathsf{OS}(e'')$.

But then $\mathsf{OS}(e) \subseteq \mathsf{OS}(e'')$ or $\mathsf{OS}(e'') \subseteq \mathsf{OS}(e)$, and thus e and e'' are not conflicting by Lemma 4.13, a contradiction to the existence of P.

Proof (of Theorem 1.4). It follows from the above discussion that the constructed coloring is a very strong rainbow coloring of G. Moreover, it uses $\mathbf{vsrc}(G)$ colors. Clearly, the coloring can be computed in polynomial time. $\square$

5 Other Algorithmic Results

In this section, we first show that 2-Vsrc can be solved in polynomial time. Then we show that for k-Vsrc is fixed parameter tractable when parameterized by $k + \mathbf{tw}(G)$, where $\mathbf{tw}(G)$ denotes the treewidth of G.

For proving both the results, we use an auxiliary graph G' defined as follows: add a vertex v_e to G' for each edge e in G; add an edge between vertices v_{e_1} and v_{e_2} in G' if and only if edges e_1 and e_2 appear together in some shortest path of G. The latter condition can be easily checked in polynomial time. Observe that $\mathbf{vsrc}(G) \leq k$ if and only if G' admits a proper k-coloring. Since 2-Coloring is solvable in polynomial time, this implies that 2-Vsrc is polynomial time solvable and hence we have proved Proposition 1.3.

It is worth noting that the chromatic number of the auxiliary graph G' constructed in the above proof always corresponds to the very strong rainbow connection number of G. However, in the transformation from G to G', we lose a significant amount of structural information. For example, if G is a path or a star ($\mathbf{tw}(G) = 1$), then G' is a clique ($\mathbf{tw}(G') = |V(G'| - 1 = |V(G)| - 2$), where we use $\mathbf{tw}(G)$ to denote the treewidth of G. However, if $\mathbf{vsrc}(G) \leq k$, then we can prove that $|V(G')| \leq k^{(k+1)} \cdot (\mathbf{tw}(G) + 1)^{(k+1)}$ as shown below.

Lemma 5.1. *Let G be any connected graph and let $\mathbf{vsrc}(G) \leq k$ and $\mathbf{tw}(G) \leq t - 1$. Then $\Delta(G) \leq kt$ and $|V(G)| \leq (kt)^k$.*

Proof. By Lemma 2.6, the fact $\mathbf{vsrc}(G) \leq k$ implies that G is k-perfectly groupable. Hence, the neighborhood of each vertex can be partitioned into k or fewer cliques. Since $\mathbf{tw}(G) \leq t - 1$, each clique of G has size at most t [21]. Hence, $\Delta(G) \leq kt$. Now observe that $\mathbf{vsrc}(G) \leq k$ implies that the diameter of G is at most k. Combined, these two facts imply that $|V(G)| \leq (kt)^k$. $\square$

Proof (of Theorem 1.5). Again, let $\mathbf{vsrc}(G) \leq k$ and $\mathbf{tw}(G) \leq t - 1$. We now construct the auxiliary graph G' as above. Now, we only need to compute the chromatic number of G'. We aim to use the algorithm by Björklund et al. [3] which computes the chromatic number of a graph on n vertices in $2^n n^{\mathcal{O}(1)}$ time. To bound $|V(G')|$, we observe that by Lemma 5.1, $|V(G)| \leq (kt)^k$ and $\Delta(G) \leq kt$. Hence, $|V(G')| = |E(G)| \leq (kt)^{(k+1)}$. Therefore, the chromatic number of G', and thereby $\mathbf{vsrc}(G)$, can be determined in $\mathcal{O}(2^{(kt)^{(k+1)}}(kt)^{\mathcal{O}(k+1)})$ time. $\square$

References

1. Ananth, P., Nasre, M., Sarpatwar, K.K.: Rainbow connectivity: Hardness and tractability. In: Chakraborty, S., Kumar, A. (eds.) Proceedings of FSTTCS 2011. LIPIcs, vol. 13, pp. 241–251. Schloss Dagstuhl (2011)
2. Basavaraju, M., Chandran, L.S., Rajendraprasad, D., Ramaswamy, A.: Rainbow connection number and radius. Graphs Comb. **30**(2), 275–285 (2014)
3. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. **39**, 546–563 (2009)
4. Chakraborty, S., Fischer, E., Matsliah, A., Yuster, R.: Hardness and algorithms for rainbow connection. J. Comb. Optim. **21**(3), 330–347 (2011)
5. Chandran, L.S., Das, A., Rajendraprasad, D., Varma, N.M.: Rainbow connection number and connected dominating sets. J. Graph Theory **71**(2), 206–218 (2012)
6. Chandran, L.S., Rajendraprasad, D.: Rainbow colouring of split and threshold graphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 181–192. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32241-9_16
7. Chandran, L.S., Rajendraprasad, D.: Inapproximability of rainbow colouring. In: Seth, A., Vishnoi, N.K. (eds.) Proceedings of FSTTCS 2013. LIPIcs, vol. 24, pp. 153–162. Schloss Dagstuhl (2013)
8. Chartrand, G., Johns, G.L., McKeon, K.A., Zhang, P.: Rainbow connection in graphs. Math. Bohem. **133**(1), 85–98 (2008)
9. Eiben, E., Ganian, R., Lauri, J.: On the complexity of rainbow coloring problems. Discret. Appl. Math. (2016, in press). https://doi.org/10.1016/j.dam.2016.10.021
10. Erdős, P., Goodman, A.W., Pósa, L.: The representation of a graph by set intersections. Canad. J. Math **18**(106–112), 86 (1966)
11. Gary, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
12. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. JCTB **16**(1), 47–56 (1974)
13. Kammer, F., Tholey, T.: Approximation algorithms for intersection graphs. Algorithmica **68**(2), 312–336 (2014)
14. Keranen, M., Lauri, J.: Computing minimum rainbow and strong rainbow colorings of block graphs. arXiv preprint arXiv:1405.6893 (2014)
15. Kowalik, Ł., Lauri, J., Socala, A.: On the fine-grained complexity of rainbow coloring. In: Sankowski, P., Zaroliagis, C.D. (eds.) Proceedings of ESA 2016. LIPIcs, vol. 57, pp. 58:1–58:16. Schloss Dagstuhl (2016)
16. Lauri, J.: Chasing the Rainbow Connection: Hardness, Algorithms, and Bounds, vol. 1428. Tampere University of Technology Publication, Tampere (2016)
17. Li, X., Shi, Y., Sun, Y.: Rainbow connections of graphs: a survey. Graphs Comb. **29**(1), 1–38 (2013)
18. Li, X., Sun, Y.: Rainbow Connections of Graphs. Springer Science & Business Media, Boston (2012). https://doi.org/10.1007/978-1-4614-3119-0
19. Li, X., Sun, Y.: An updated survey on rainbow connections of graphs - a dynamic survey. Theory Appl. Graphs **0**, 3 (2017)
20. Roberts, F.S.: Applications of edge coverings by cliques. Discret. Appl. Math. **10**(1), 93–109 (1985)
21. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. JCTB **7**, 309–322 (1986)

22. Uchizawa, K., Aoki, T., Ito, T., Suzuki, A., Zhou, X.: On the rainbow connectivity of graphs: complexity and FPT algorithms. Algorithmica **67**(2), 161–179 (2013)
23. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of STOC 2006, pp. 681–690. ACM (2006)

New Integer Linear Programming Models for the Vertex Coloring Problem

Adalat Jabrayilov(✉) and Petra Mutzel

Department of Computer Science, TU Dortmund University, Dortmund, Germany
{adalat.jabrayilov,petra.mutzel}@tu-dortmund.de

Abstract. The vertex coloring problem asks for the minimum number of colors that can be assigned to the vertices of a given graph such that each two neighbors have different colors. The problem is NP-hard. Here, we introduce new integer linear programming formulations based on partial-ordering. They have the advantage that they are as simple to work with as the classical assignment formulation, since they can be fed directly into a standard integer linear programming solver. We evaluate our new models using Gurobi and show that our new simple approach is a good alternative to the best state-of-the-art approaches for the vertex coloring problem. In our computational experiments, we compare our formulations with the classical assignment formulation and the representatives formulation on a large set of benchmark graphs as well as randomly generated graphs of varying size and density. The evaluation shows that the partial-ordering based models dominate both formulations for sparse graphs, while the representatives formulation is the best for dense graphs.

Keywords: Graph coloring · Vertex coloring
Integer linear programming

1 Introduction

The vertex coloring problem (VCP) belongs to the classical optimization problems. This problem asks for the minimum number of colors, which are assigned to the vertices of a graph such that no two adjacent vertices get the same color. The minimum number of colors is called *chromatic number* and denoted by χ. Computing the chromatic number of a graph is NP-hard [9]. Since the vertex coloring problem has many applications, e.g., register allocation, scheduling, frequency assignment and timetabling, there is a vast amount of literature on this problem (see, e.g., [18] for a survey). However, in contrast to other classical combinatorial optimization problems such as the Travelling Salesman Problem, where large instances can be solved to optimality, this is not true for the VCP.

There are two main lines of research based on integer linear programming (ILP) formulations for VCP. The natural formulation introduces binary variables

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 640–652, 2018.
https://doi.org/10.1007/978-3-319-77404-6_47

that assign colors to vertices. A vertex v in the graph $G = (V, E)$ is assigned color i iff the corresponding binary variable $x_{v,i}$ gets value 1. This *assignment formulation* has the advantage that it is simple and easy to use. Since the number of variables and constraints of this ILP model is polynomial in $|V|$, it can be fed directly into a (commercial) integer linear programming solver such as *SCIP* [1], *LP_Solve*, *Cplex*, or *Gurobi*. Due to the inherent symmetry in the model (the colors are not distinguishable) only small instances can be solved to optimality. However, additional constraints can be added by which the symmetry can be reduced. The second approach has been suggested by Mehrotra and Trick [19] and is based on the observation that each color class defines an independent set (no two vertices in the set are adjacent) in the graph. The variables correspond to independent sets and the ILP model searches for the minimum number of these independent sets that cover the graph. Since the number of independent sets can be of exponential size, the solution approach is based on column generation. Solution algorithms based on this *Set Covering Formulation* are of complex nature. Mehrotra and Trick [19] suggest a branch-and-price algorithm for solving the ILP model. Both ILP formulations have been studied and improved by additional ideas in the literature leading to complex branch-and-cut algorithms using additional classes of constraints, special branching schemes, separation procedures, and special procedures for providing good upper bounds. The computational studies in the literature show that none of the ILP models dominates the other one.

ILP formulations based on partial-ordering have shown to be practically successful in the area of graph drawing [14]. Here, we suggest a new ILP formulation based on partial-ordering for the vertex coloring problem. It has the advantage that it is as simple to work with as the assignment formulation, since it is of polynomial size and can be fed directly into a standard integer linear programming solver. We further suggest a hybrid ILP formulation which combines the advantages of the assignment formulation with those of the partial-ordering model.

We evaluate the new models using the ILP solver *Gurobi* and show that our new simple approaches dominate the assignment formulation on the tested benchmark sets and are a good alternative to the best state-of-the-art approaches for the vertex coloring problem. We also present the first experimental comparison with the representatives formulation which has been suggested by Campelo et al. [3,4]. Since it introduces variables for every pair of non-adjacent vertices, this formulation seems to be advantageous for dense graphs. Our computational results support this observation.

2 State-of-the-Art

Eppstein [8] has shown that it is possible to solve the vertex coloring problem by enumerating all maximal independent sets in the graph in time $O((4/3 + 3^{4/3}/4)^{|V|})$ which is about $2.4150^{|V|}$. In practice, the successful approaches are much faster than that. There are two main directions followed by exact algorithms based on ILP models for the problem: the ILP-based assignment model

(Sect. 2), and the ILP-based set covering formulation. There are also a lot of experimental evaluations of these methods. Altogether they have not shown a superiority of one of these lines of research. From those, the assignment model is the simplest one, since it can directly be fed into a standard ILP-solver. Another simple ILP formulation is the so-called representatives ILP model. It seems that there is no experimental evaluation concerning this model. In the literature there also exist alternative ILP models and alternative approaches (e.g., based on Constraint Programming [11]). However, the aim of this work is to concentrate on simple ILP formulations that are competitive with the best state-of-the-art approaches. There is also a vast literature on heuristic approaches. For a detailed overview of heuristic and exact approaches, see the survey by Malaguti and Toth [18] and Burke et al. [2].

Assignment-Based ILP Model. The classical ILP model for a graph $G = (V, E)$ is based on assigning color i to vertex $v \in V$. For this, the assignment variables $x_{v,i}$ are defined for each vertex v and color $i \in \{1, \ldots, H\}$ with $x_{v,i} = 1$ if vertex v is assigned to color i and $x_{v,i} = 0$ otherwise. H is an upper bound of the number of colors (e.g., the result of a heuristic) and is at most $|V|$. For modelling the objective function, an additional binary variable w_i for each $i \in \{1, \ldots, H\}$ is needed which gets value 1 iff color i is used in the coloring. The model is given by:

$$\text{(ASS-S)} \min \textstyle\sum_{1 \le i \le H} w_i \tag{1}$$

$$\text{s.t.} \quad \textstyle\sum_{i=1}^{H} x_{v,i} = 1 \quad \forall v \in V \tag{2}$$

$$x_{u,i} + x_{v,i} \le w_i \ \forall (u, v) \in E, \ i = 1, \ldots, H \tag{3}$$

$$x_{v,i}, w_i \in \{0, 1\} \ \forall v \in V, \ i = 1, \ldots, H \tag{4}$$

The objective function minimizes the number of used colors. Equations (2) ensure that each vertex receives exactly one color. For each edge there is a constraint (3) making sure that adjacent vertices receive different colors. This model has the advantage that it is simple and easy to use. It can be easily extended to generalizations and/or restricted variants of the graph coloring problem. Since the number of variables is quadratic in $|V|$ (bounded by $H(|V| + 1)$) and the number of constraints is cubic in $|V|$ (exactly $|V| + H|E| = O(|V||E|)$ constraints of type 2 and 3), it can directly be used as input for a standard ILP solver. The main drawback of this model is the inherent symmetry, since there are $\binom{H}{\chi}$ possibilities to select χ from H colors thus leading to exponentially (in the number of colors) many equivalent solutions. In order to remove this type of symmetry, Mendez-Diaz and Zabala [20,21] suggest to extend (ASS-S) through the following set of constraints:

$$w_i \le \sum_{v \in V} x_{v,i} \qquad i = 1, \ldots, H \tag{5}$$

$$w_i \le w_{i-1} \qquad i = 2, \ldots, H \tag{6}$$

We call this extended model (ASS). These constraints ensure that the color i is only assigned to some vertex, if color $i-1$ is already assigned to another one. Moreover, they present several sets of constraints that arose from their studies of the associated polytope. In order to solve the new strengthened ILP model, they developed a branch-and-cut algorithm.

Representatives ILP Model. A vertex coloring divides the vertices into disjoint color classes. Campêlo et al. [3,4] suggested a model in which each color class is represented by exactly one vertex. For this, they suggest to introduce a binary variable x_{uv} for each non-adjacent pair of vertices $u, v \in V$ which is 1 if and only if the color of v is represented by u. Additional binary variables x_{uu} indicate if u is the representative of its color class. Let $\bar{N}(u)$ be the set of non-adjacent vertices to u. The constraints are as follows:

$$\text{(REP)} \quad \min \textstyle\sum_{u \in V} x_{uu} \tag{7}$$

$$\textstyle\sum_{u \in \bar{N}(v) \cup v} x_{uv} \geq 1 \;\; \forall v \in V \tag{8}$$

$$x_{uv} + x_{uw} \leq x_{uu} \quad \forall u \in V, \; \forall e = (v, w) \in G[\bar{N}(u)] \tag{9}$$

$$x_{uv} \in \{0, 1\} \qquad \forall \text{ non-adjacent vertex pairs } \; u, v \;\; \text{or} \;\; u = v \tag{10}$$

Inequalities (8) require that for any vertex $v \in V$, there must exist a representative which can be v itself or some vertex from $\bar{N}(v)$. Inequalities (9) state that a vertex u cannot be the representative for both endpoints of an edge (v, w) and that in the case that $x_{uv} = 1$ (u is the representative of vertex v) also the variable x_{uu} must take value 1. The advantage of this model is its simplicity and its compactness. It has exactly $|\bar{E}| + |V|$ variables and up to $|V| + |V||E|$ many constraints, where $\bar{E}$ is the set of non-adjacent vertex pairs of $G = (V, E)$, i.e., $\bar{E} = (V \times V) \setminus E$. With growing density of the graphs, the number of constraints increases but the number of variables decreases and converges towards $|V|$. In [3], Campelo et al. mention that the symmetry in this model may lead to problems with branch-and-bound based solvers. The reason for this lies in the fact that within a color class any of the vertices in this class can be the representative. In order to circumvent this, the authors define an ordering on the vertices and require that in each color class only the vertex with the smallest number is allowed to be the representative of this class. The ILP model arising from this requirement is called the *asymmetric representatives formulation* (AREP). The authors [3] study the polytopes associated with both representative formulations. They suggest to add constraints based on cliques, odd-holes, anti-holes, wheels, and independent sets in order to strengthen the model. Moreover, they provide a comparison with the set covering based formulation. In [5], Campos et al. study the asymmetric representatives formulation and the corresponding polytope. Their results lead to complete characterizations of the associated polytopes for some specific graph classes. Up to our knowledge, no computational experiments have been published in the literature.

3 Partial-Ordering Based ILP Models

3.1 A Pure Partial-Ordering Based ILP Model: POP

Our new binary model considers the vertex coloring problem as a partial-ordering problem (POP). We assume that the H colors $(1, \dots, H)$ are linearly ordered. Instead of directly assigning a color to the vertices, we determine a partial order of the union of the vertex set and the set of ordered colors. For this, we determine the relative order of each vertex with respect to each color in the color ordering. More specific: for every color i and every vertex $v \in V$ our variables provide the information if v is smaller or larger than i. We denote these relations by $v \prec i$ or $v \succ i$, resp. In other words, the colors and the vertices build a partially ordered set in which all pairs of the form (v, i) with $v \in V, i = 1, \dots, H$ are comparable. We define the following POP variables:

$$\forall v \in V,\ i = 1, \dots, H: \quad y_{i,v} = \begin{cases} 1 & v \succ i \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall v \in V,\ i = 1, \dots, H: \quad z_{v,i} = \begin{cases} 1 & v \prec i \\ 0 & \text{otherwise.} \end{cases}$$

If vertex v has been assigned to color i, then v is neither smaller nor larger than i and we have $y_{i,v} = z_{v,i} = 0$. The connection with the assignment variables x from the (ASS) model is as follows:

$$x_{v,i} = 1 - (y_{i,v} + z_{v,i}) \qquad \forall v \in V,\ i = 1, \dots, H \tag{11}$$

We select an arbitrary vertex $q \in V$ and formulate our new binary program:

$$\begin{aligned}
\text{(POP)} \quad & \min 1 + \textstyle\sum_{1 \le i \le H} y_{i,q} & & (12)\\
\text{s.t.} \quad & z_{v,1} = 0 & \forall v \in V & \quad (13)\\
& y_{H,v} = 0 & \forall v \in V & \quad (14)\\
& y_{i,v} - y_{i+1,v} \ge 0 & \forall v \in V,\ i = 1, \dots, H-1 & \quad (15)\\
& y_{i,v} + z_{v,i+1} = 1 & \forall v \in V,\ i = 1, \dots, H-1 & \quad (16)\\
& y_{i,u} + z_{u,i} + y_{i,v} + z_{v,i} \ge 1 & \forall (u,v) \in E,\ i = 1, \dots, H & \quad (17)\\
& y_{i,q} - y_{i,v} \ge 0 & \forall v \in V,\ i = 1, \dots, H-1 & \quad (18)\\
& y_{i,v}, z_{v,i} \in \{0,1\} & \forall v \in V,\ i = 1, \dots, H & \quad (19)
\end{aligned}$$

Lemma 1. *The integer linear programming formulation (POP) is correct: Any feasible solution of the ILP corresponds to a feasible vertex coloring and the value of the objective function corresponds to the chromatic number of G.*

Proof. All original vertices need to be embedded between the colors 1 and H. Constraints (13) and (14) take care of this. By transitivity, a vertex that is larger than color $i+1$ is also larger than i (constraints (15)). Constraints (16) express that each vertex v is either larger than i (i.e. $y_{i,v} = 1$) or smaller than $i + 1$ and not both. These constraints jointly with constraints (15) ensure that each vertex will be assigned to exactly one color, i.e. there is no color pair $i \neq j$ with $y_{i,v} = z_{v,i} = 0$ and $y_{j,v} = z_{v,j} = 0$. We show this by contradiction. Let $y_{i,v} = z_{v,i} = 0$. In the case $j < i$, as $z_{v,i} = 0$ we have $y_{i-1,v} = 1$ by (16). Therefore we have $y_{j,v} = 1$ for each $j \leq i - 1$ by (15) which is a contradiction to $y_{j,v} = 0$. In the case $j > i$, as $y_{i,v} = 0$ we have $y_{k,v} = 0$ for each $k \geq i$ by (15). Therefore we have $z_{v,k+1} = 1$ by (16) leading to $z_{v,j} = 1$ for each $j \geq i + 1$ which is a contradiction to $z_{v,j} = 0$. Constraints (17) prevent assigning the same color i to two adjacent vertices u and v. Constraints (18) take care of the fact that our chosen vertex q will be assigned to the largest chosen color. So if q is not larger than color i then this will be true for all other vertices $v \in V \setminus \{q\}$. Because of this constraint, the objective function indeed minimizes the number of assigned colors since it sums up the number of colors smaller than q. In order to get the number of chosen colors, we need to add one for the color assigned to q. We say that q *represents the chromatic number* of G.

Comparison with the assignment model (POP) has $2 \cdot H|V|$ binary variables and about $4|V|H + 2|V| + |E|H$ constraints. Notice that the Eqs. (13), (14) and (16) can be used to eliminate $(H + 1)|V|$ variables. Hence the reduced model has $(H - 1) \cdot |V|$ variables, and thus $H + V$ variables less than (ASS), which has $H(|V| + 1)$ variables.

Mendez-Diaz and Zabala [20] mention that the classical branching rule (to branch on fractional assignment variables by setting them to 1 in one subproblem and to 0 in another subproblem) produces quite unbalanced enumeration trees. This is the case because of setting $x_{v,i} = 1$ implies $x_{v,j} = 0$ for all $j \neq i$, while setting $x_{v,i} = 0$ does not provide any further information. The model (POP) does not have this problem, since setting a POP-variable $y_{i,v} = 0$ implies $y_{j,v} = 0$ for all j with $j > i$ and setting $y_{i,v} = 1$ implies $y_{j,v} = 1$ for all j with $j < i$ because of (15).

As already discussed, the original (ASS-S) model has inherent symmetries, which can be resolved by additional constraints leading to the (ASS) model. In the new (POP) model, this type of symmetry does not occur.

3.2 A Hybrid Partial-Ordering Based ILP Model: POP2

Our second ILP formulation is a slight modification of the first model and can be seen as a hybrid of the models (POP) and (ASS). It is the consequence of the observation that with growing density the (POP) constraint matrix contains more nonzero elements than the (ASS) constraint matrix. This is due to the constraints (17), which are responsible for the valid coloring of adjacent vertices, and contain twice as many nonzero coefficients as the corresponding (ASS) constraints (3). Therefore, we substitute (17) by (11) and the following constraints:

$$x_{u,i} + x_{v,i} \leq 1 \qquad \forall (u, v) \in E,\ i = 1, \ldots, H. \tag{20}$$

This reduces the number of nonzero coefficients from $4|E|H$ to $2|E|H + 3|V|H$ giving a reduction ratio of about two in dense graphs. Although we added $|V|H$

new assignment variables, the dimension of the problem remains unchanged, since the new variables directly depend on the POP-variables by equality (11).

4 Computational Experiments

In our experiments we are interested in answering the following questions:

- (H1): Do our new partial-ordering based ILP formulations dominate the classical assignment ILP model (ASS) on a set of benchmark instances?
- (H2): Does one of the two partial-ordering models dominate the other one?
- (H3): How do the simple models behave compared to the state-of-the-art algorithms on a benchmark set of graphs?
- (H4): Does (AREP) dominate the other approaches on dense instances?

4.1 Implementation

The preprocessing techniques (a)–(d) are widely used (e.g., [11,12,17,18,20,21]):

(a) A vertex u is *dominated* by vertex v, $v \neq u$, if the neighborhood of u is a subset of the neighborhood of v. In this case, the vertex u can be deleted, the remaining graph can be colored, and at the end u can get the color of v.
(b) To reduce the number of variables we are interested in getting a small upper bound H for the number of needed colors.
(c) Since any clique represents a valid lower bound for the vertex coloring problem one can select any maximal clique and precolor it.
(d) In the case of equal lower and upper bounds the optimal value has been found, hence no ILP needs to be solved.

We extended (c) as follows:

(e) In (ASS), (POP), and (POP2) we can fix more variables if we try to find the clique Q with $\max(|Q|H + |\delta(Q)|)$, where $\delta(Q) := \{(u,v) \in E\colon |\{u,v\} \cap Q| = 1\}$. The first term $|Q|H$ is due to the fact that we can fix H variables for each vertex in Q. After precoloring of some vertex $u \in Q$, the neighbors v of u cannot receive the same color as u, e.g. if the assignment variable $x_{u,i} = 1$ then $x_{v,i} = 0$. Hence we can fix one variable for each edge $(u,v) \in \delta(Q)$.

To represent the chromatic number in (POP) and (POP2), we pick any vertex q from the clique Q found in the preprocessing. The remaining vertices from the clique are precolored with colors $1, \cdots, |Q| - 1$.

We have implemented the simple models (ASS), (POP), (POP2), (REP) and (AREP) using the Gurobi-python API. To find large maximal cliques in (c) and (e), we used a heuristic from the python library http://networkx.readthedocs.io. The source codes are available on our benchmark site[1]. As mentioned in Subsect. 3.1, in our implementation of (POP) and (POP2) we used (13), (14) and (16) and eliminated all z variables.

[1] https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks.

4.2 Test Setup and Benchmark Set of Graphs

To solve the models, we used the Gurobi 6.5 single-threadedly on Intel Xeon E5-2640, 2.60 GHz, with 64 GB of memory and running Ubuntu Linux 14.04. The `dfmax` benchmark [7], which is used in the DIMACS challenge to compare the results obtained with different machines, needs on our machine 5.54 s for `r500.5`.

We considered two benchmark sets, which are available at [15]. To compare the new approach with the state-of-the-art algorithms from the literature, we used subsets of the DIMACS [24] benchmark sets. From 119 DIMACS graphs we have chosen the hard instances according to the Google benchmark site [10] and the `GPIA` graphs, which are obtained from a matrix partitioning problem to determine sparse Jacobian matrices. The second benchmark set consists of 340 randomly generated graphs $G(n, p)$, which have n vertices and an edge between each vertex pair with probability p. This set contains also two subsets:

set70: 100 instances: 20 graphs with $n = 70$ and $p = 0.1, 0.3, 0.5, 0.7, 0.9$.
sparse100: 240 instances: 20 graphs for each $n = 80, 90, 100$ and for each $p = 0.1, 0.15, 0.2, 0.25$.

4.3 Experimental Evaluation

Table 1 shows the results for the DIMACS benchmark set. The new approach is compared with the assignment and the representatives models and with other state-of-the-art algorithms [6,17,20,21] from the literature. The table contains 37 of the 68 hard instances, which can be solved by at least one of the considered algorithms. Columns 1–3 show the instance names and sizes. Column 4 describes the hardness of the instances according to the Google site [10]. Columns 5–16 display the lower and upper bounds as well as the running times of the simple models (AREP), (POP), (POP2) and (ASS) that have been obtained within a time limit of one hour by the ILP-solver. An entry tl indicates that the time limit is reached. The times are provided in seconds for solving the reduced ILPs after prepocessing. The preprocessing is done with python and took a few seconds for most instances and not more than one minute. Columns 5–7 show the results of (AREP) with preprocessings (a)–(d). (AREP) does not need (b) directly, but indirectly due to (d). Columns 8–16 show appropriate results for (POP), (POP2) and (ASS). Since (e) can reduce the number of assignment and POP variables, we implemented (ASS), (POP) and (POP2) also with (e) instead of (c). While both versions (ASS)+(c) and (ASS)+(e) solved 21 instances, the average runtime of solved instances by (ASS)+(e) (171.42 s) was smaller than (ASS)+(c) (405.02 s). Also for (POP) and (POP2) it turned out that (e) is better than (c). Table 1 displays the results of (POP), (POP2), (ASS) corresponding to (e) only (due to space restrictions). Since all the solved instances in the recent paper [6] and by (REP) can be solved by (AREP), the table displays the results of the latter only. Note, that the instance `4-Insertions_3` (taking 8 h by [6]) exceeds the 1 h time limit taking 3641 s by (AREP).

Table 1. Results for the hard instances from DIMACS benchmark set

instance	|V|	|E|	class	AREP			POP			POP2			ASS+(e)			[20]	[21]	[17]
				lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time			
1-FullIns_4	93	593	NP-m	5.0	5.0	0.57	5.0	5.0	0.01	5.0	5.0	0.02	5.0	5.0	0.01	0.1		tl
1-FullIns_5	282	3247	NP-?	6.0	6.0	279.46	6.0	6.0	7.42	6.0	6.0	2.04	6.0	6.0	2.71	tl		tl
2-FullIns_4	212	1621	NP-m	6.0	6.0	0.64	6.0	6.0	0.04	6.0	6.0	0.03	6.0	6.0	0.02	tl	4	tl
2-FullIns_5	852	12201	NP-?	6.0	7.0	tl	7.0	7.0	6.19	7.0	7.0	86.48	7.0	7.0	17.66	tl		tl
3-FullIns_3	80	346	NP-m	6.0	6.0	0.01	6.0	6.0	0.00	6.0	6.0	0.00	6.0	6.0	0.00	0.1		2.9
3-FullIns_4	405	3524	NP-?	7.0	7.0	2.07	7.0	7.0	0.05	7.0	7.0	0.04	7.0	7.0	0.05	tl		tl
3-FullIns_5	2030	33751		7.0	8.0	tl	8.0	8.0	15.19	8.0	8.0	39.31	8.0	8.0	32.15	tl		tl
4-FullIns_3	114	541	NP-m	7.0	7.0	0.01	7.0	7.0	0.01	7.0	7.0	0.01	7.0	7.0	0.00	3		3.4
4-FullIns_4	690	6650	NP-?	8.0	8.0	1.27	8.0	8.0	0.08	8.0	8.0	0.04	8.0	8.0	0.04	tl		tl
4-FullIns_5	4146	77305		7.0	9.0	tl	9.0	9.0	11.65	9.0	9.0	19.22	9.0	9.0	93.98	tl		tl
4-Insertions_3	79	156	NP-m	3.0	4.0	tl	4.0	4.0	11.71	4.0	4.0	19.18	4.0	4.0	64.10	4204		tl
5-FullIns_3	154	792	NP-m	8.0	8.0	0.01	8.0	8.0	0.01	8.0	8.0	0.01	8.0	8.0	0.00	20		4.6
5-FullIns_4	1085	11395	NP-?	9.0	9.0	3.73	9.0	9.0	0.08	9.0	9.0	0.08	9.0	9.0	0.06	tl		tl
ash608GPIA	1216	7844	NP-m	$-\infty$	$+\infty$	tl	4.0	4.0	42.50	4.0	4.0	62.42	4.0	4.0	854.60	692		2814.8
ash958GPIA	1916	12506	NP-m	$-\infty$	$+\infty$	tl	4.0	4.0	109.15	4.0	4.0	122.92	4.0	6.0	tl	tl	4236	tl
DSJC125.5	125	3891	NP-h	14.0	20.0	tl	11.0	20.0	tl	13.0	22.0	tl	13.0	21.0	tl	tl		18050.8
DSJC125.9	125	6961	NP-h	44.0	44.0	1.13	35.0	50.0	tl	42.0	44.0	tl	42.0	45.0	tl	tl		3896.9
DSJC250.9	250	27897	NP-h	72.0	72.0	1367.70	39.0	$+\infty$	tl	45.0	$+\infty$	tl	40.0	89.0	tl	tl		tl
DSJR500.1c	500	121275	NP-h	85.0	85.0	0.10	77.0	$+\infty$	tl	83.0	86.0	tl	78.0	$+\infty$	tl	tl		288.5
DSJR500.5	500	58862	NP-h	$-\infty$	$+\infty$	tl	115.0	$+\infty$	tl	122.0	122.0	551.92	115.0	$+\infty$	tl	tl		342.2
le450_15a	450	8168	NP-m	15.0	21.0	tl	15.0	16.0	tl	15.0	15.0	690.84	15.0	15.0	963.46	tl		0.4
le450_15b	450	8169	NP-?	15.0	19.0	tl	15.0	15.0	3473.07	15.0	15.0	827.73	15.0	15.0	1283.05	tl		0.2
le450_15c	450	16680	NP-?	$-\infty$	450.0	tl	15.0	$+\infty$	tl	15.0	$+\infty$	tl	15.0	25.0	tl	tl		3.1
le450_15d	450	16750	NP-?	$-\infty$	450.0	tl	15.0	26.0	tl	15.0	26.0	tl	15.0	$+\infty$	tl	tl		3.8
le450_25c	450	17343	NP-?	$-\infty$	450.0	tl	25.0	30.0	tl	25.0	31.0	tl	25.0	$+\infty$	tl	tl		1356.6
le450_25d	450	17425	NP-?	25.0	450.0	tl	25.0	30.0	tl	25.0	31.0	tl	25.0	$+\infty$	tl	tl		66.6
le450_5a	450	5714	NP-?	$-\infty$	450.0	tl	5.0	9.0	tl	5.0	5.0	25.65	5.0	5.0	62.12	tl		0.3
le450_5b	450	5734	NP-?	$-\infty$	450.0	tl	5.0	8.0	tl	5.0	5.0	171.03	5.0	5.0	194.73	tl		0.2
mug100_1	100	166	NP-m	4.0	4.0	0.62	4.0	4.0	0.38	4.0	4.0	0.13	4.0	4.0	0.20	60		14.4
mug100_25	100	166	NP-m	4.0	4.0	0.71	4.0	4.0	0.72	4.0	4.0	0.48	4.0	4.0	0.49	60		12
qg.order40	1600	62400	NP-m	$-\infty$	$+\infty$	tl	40.0	45.0	tl	40.0	40.0	594.96	40.0	46.0	tl	tl		2.9
qg.order60	3600	212400	NP-?	$-\infty$	$+\infty$	tl	60.0	68.0	tl	60.0	62.0	tl	$-\infty$	68.0	tl	tl		3.8
queen10_10	100	1470	NP-h	10.0	12.0	tl	10.0	12.0	tl	10.0	12.0	tl	10.0	12.0	tl	tl		686.9
queen11_11	121	1980	NP-h	11.0	13.0	tl	11.0	13.0	tl	11.0	13.0	tl	11.0	13.0	tl	tl		1865.7
school1_nsh	352	14612	NP-m	14.0	14.0	891.92	14.0	14.0	24.66	14.0	14.0	13.25	14.0	14.0	30.38	0		17
wap05a	905	43081	NP-m	$-\infty$	$+\infty$	tl	40.0	$+\infty$	tl	50.0	50.0	1317.36	41.0	$+\infty$	tl	tl		293.2
wap06a	947	43571	NP-?	$-\infty$	$+\infty$	tl	40.0	$+\infty$	tl	40.0	$+\infty$	tl	40.0	$+\infty$	tl	tl		175
solved:						15			19			**25**			21		9+2	**25**
avg. time						170.00			194.89			182.36			171.42		843.6	1196.3

Columns 17 and 18 are taken from [20,21] and show the running times of two (ASS)-based branch-and-cut algorithms suggested by Mendez-Diaz and Zabala. Column 18 ([21]) contains only the additional solved instances, i.e. the instances which have not been solved in [20]. Column 19 is taken from [17] and shows the running times of a set-covering-based branch-and-price algorithm suggested by Malaguti et al. [17]. Notice that the comparison of running times is not quite fair, since [21] and [17] report that their machines need 24 s and 7 s, respectively, for the benchmark [7] instance r500.5, while our machine needs 5.54 s. However, [20,21] and [17] used 2 h and 10 h as time limit respectively, while we only used 1 h. Nevertheless, it is interesting to see the number of optimal solved instances by each algorithm. This is displayed in the row *"solved"*. The table shows the average time for optimally solved instances in the last row.

We can see that the model (POP2) as well as the set-covering-based approach [17] have solved the highest number of instances (25 out of 68) to provable optimality. Although (POP2) was run on an approximately 1.26 ($= 7s/5.54\,\text{s}$) times faster machine its average runtime is about 6.56 ($= 1196.3\,\text{s}/182.36\,\text{s}$) times faster than that described in [17]. The models (ASS)+(e), (POP), (AREP), (REP) have solved 21, 19, 15, 13 instances respectively, while the algorithms [20,21] only solved 11 ($= 9 + 2$) instances. It seems that the hybrid model (POP2)

Table 2. Results of the simple models for the `GPIA` graphs from DIMACS set

				Time	AREP			POP			POP2			ASS+(c)			ASS+(e)				
instance	$\|V\|$	$\|E\|$	class	limit	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	old lb	old ub
will199GPIA	701	6772	NP-s	1h	$-\infty$	$+\infty$	tl	7	7	6.68	7	7	11.35	7	7	6.98	7	7	7.24		
ash331GPIA	662	4181	NP-s	1h	$-\infty$	$+\infty$	tl	4	4	11.94	4	4	16.07	4	4	3.45	4	4	74.46		
ash608GPIA	1216	7844	NP-m	1h	$-\infty$	1215	tl	4	4	43.06	4	4	62.98	4	4	607.82	4	4	855.11		
ash958GPIA	1916	12506	NP-m	1h	$-\infty$	$+\infty$	tl	4	4	108.57	4	4	124.75	4	6	tl	4	6	tl		
abb313GPIA	1555	53356	NP-?	3h	$-\infty$	853	tl	8	10	tl	**9**	10	tl	8	12	tl	8	11	tl	8[20]	9[17]

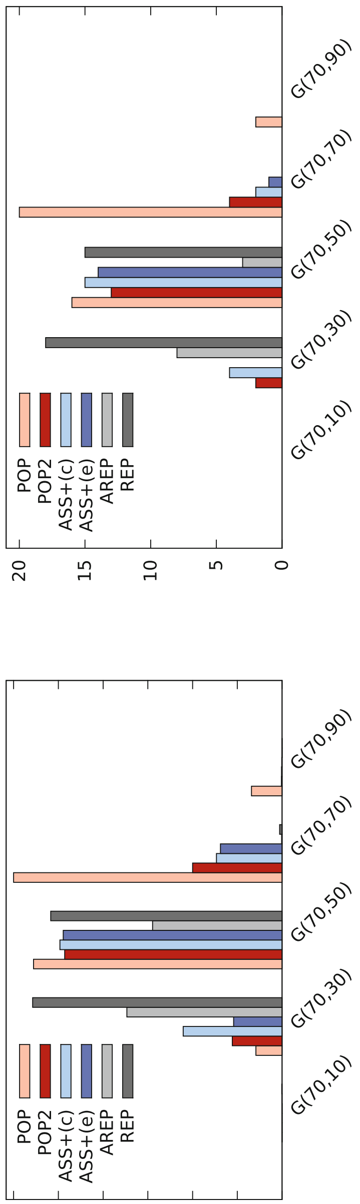
combines the advantages of the pure (POP) model and assignment models. A closer look at the single instances shows that it solved all instances which are solved by (POP) or (ASS). As indicated in our theoretical analysis in Sect. 3.2 the hybrid model dominates the pure model for denser instances.

Table 2 shows the results for the simple models for all five DIMACS `GPIA` graphs as well as old lower and upper bounds for the instance `abb313GPIA`. Since (POP) and (POP2) solved all `GPIA` graphs except `abb313GPIA` within a time limit of 1 h, we decided to increase the time limit for this graph to 3 h. (POP2) achieved a lower bound of 9 in 7769 sec thus improving the best lower bound found. To our knowledge, this new model is the first one solving all of the remaining four `GPIA` graphs.

In order to study the behaviour of the implemented simple models for instances with varying size and density, we first used the benchmark *set70*, for which the results are displayed in Fig. 1. Figure 1(a) shows the average runtime for each set $G(70, p)$ for each density $p = 10, 30, 50, 70, 90$, while Fig. 1(b) shows the number of unsolved instances for each set. (POP) and (ASS)+(e) were able to solve all instances of densities 10 and 30, where the average runtime of (POP) is about 2 times smaller than that of (ASS)+(e). Also here the model (ASS) with the new preprocessing technique (e) is significantly faster than with (c). From density 50 on, the (POP) model seems to have more problems than the other models. The (ASS)+(c), (ASS)+(e) and the (POP2) model deliver similar quality with similar running times for the denser graphs. From density 50 on, the (AREP) model clearly dominates all other models. Since (POP) was the fastest model on the sparse graphs of *set70*, we decided to evaluate larger sparse graphs and generated the set *sparse100*. The corresponding results are shown in Fig. 2. For the larger instances, both partial-ordering based models (POP) and (POP2) dominate the other models. The representative model already gets problems with the smallest instances in the set.

We conclude our work with answering our questions from the beginning:

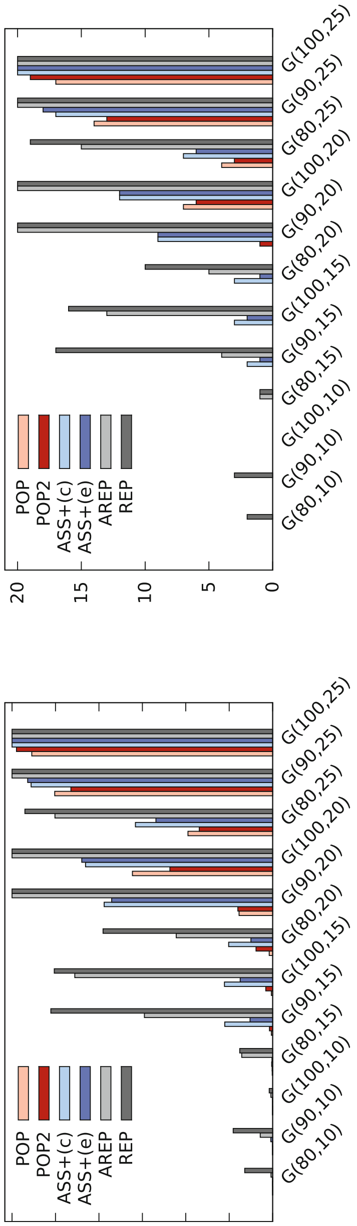
- (H1): (POP2) is able to solve more DIMACS instances to provable optimality than the assignment model. This is not true for the (POP) model. On the random graphs the models (POP) and (POP2) dominate the assignment model on graphs with density $p \leq 30$ and $p \leq 50$, respectively.
- (H2): The (POP2) model dominates the original model (POP) on the harder DIMACS instances as well as on the tested dense random graphs. The explanation lies in the fact discussed in Sect. 3.2. It seems that the (POP2) model combines the advantages of (POP) which is better for sparse graphs and (ASS) which is better for dense graphs.



(a) Average Execution Time [s]

(b) Number of unsolved instances

Fig. 1. Comparison of the simple models on the benchmark *set70*



(a) Average Execution Time [s]

(b) Number of unsolved instances

Fig. 2. Comparison of the simple models on the benchmark *sparse100*

– (H3): The simple models are able to solve very hard instances from the DIMACS benchmark set. A comparison with the computational results of the state-of-the-art algorithms (such as [12,13,16–23]) shows that the quality of the suggested algorithms is about the same (also see Tables 1 and 2). Some of the approaches are able to solve some of the instances faster, but they are slower on other instances.
– (H4): The representatives model does clearly dominate the other models on dense instances. This can be seen on the denser instances of the DIMACS graphs and on the series of random graphs with increasing density.

Acknowledgements. This work was partially supported by DFG, RTG 1855.

References

1. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: a new approach to integrate CP and MIP. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 6–20. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68155-7_4
2. Burke, E.K., Mareček, J., Parkes, A.J., Rudová, H.: A supernodal formulation of vertex colouring with applications in course timetabling. Ann. Oper. Res. **179**(1), 105–130 (2010)
3. Campêlo, M.B., Campos, V.A., Corrêa, R.C.: On the asymmetric representatives formulation for the vertex coloring problem. Discrete Appl. Math. **156**(7), 1097–1111 (2008)
4. Campêlo, M.B., Corrêa, R.C., Frota, Y.: Cliques, holes and the vertex coloring polytope. Inf. Process. Lett. **89**(4), 159–164 (2004)
5. Campos, V., Corrêa, R.C., Delle Donne, D., Marenco, J., Wagler, A.: Polyhedral studies of vertex coloring problems: The asymmetric representatives formulation. ArXiv e-prints, August 2015
6. Cornaz, D., Furini, F., Malaguti, E.: Solving vertex coloring problems as maximum weight stable set problems. Disc. Appl. Math. **217**(Part 2), 151–162 (2017)
7. Benchmarking machines and testing solutions (2002). http://mat.gsia.cmu.edu/COLOR02/BENCHMARK/benchmark.tar
8. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algorithms Appl. **7**(2), 131–140 (2003)
9. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco, CA, USA (1979)
10. Gualandi, S. Chiarandini, M.: Graph coloring instances (2017). https://sites.google.com/site/graphcoloring/vertex-coloring
11. Gualandi, S., Malucelli, F.: Exact solution of graph coloring problems via constraint programming and column generation. INFORMS J. Comput. **24**(1), 81–100 (2012)
12. Hansen, P., Labbé, M., Schindl, D.: Set covering and packing formulations of graph coloring: algorithms and first polyhedral results. Discrete Optim. **6**(2), 135–147 (2009)
13. Held, S., Cook, W., Sewell, E.: Maximum-weight stable sets and safe lower bounds for graph coloring. Math. Program. Comput. **4**(4), 363–381 (2012)
14. Jabrayilov, A., Mallach, S., Mutzel, P., Rüegg, U., von Hanxleden, R.: Compact layered drawings of general directed graphs. In: Hu, Y., Nöllenburg, M. (eds.) GD 2016. LNCS, vol. 9801, pp. 209–221. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50106-2_17

15. Jabrayilov, A., Mutzel, P. (2017). https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks
16. Johnson, D.S., Trick, M. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. AMS, Providence (1996)
17. Malaguti, E., Monaci, M., Toth, P.: An exact approach for the vertex coloring problem. Discrete Optim. **8**(2), 174–190 (2011)
18. Malaguti, E., Toth, P.: A survey on vertex coloring problems. Int. Trans. Oper. Res. **17**, 1–34 (2010)
19. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS J. Comput. **8**(4), 344–354 (1996)
20. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. Discrete Appl. Math. **154**(5), 826–847 (2006)
21. Méndez-Díaz, I., Zabala, P.: A cutting plane algorithm for graph coloring. Discrete Appl. Math. **156**(2), 159–179 (2008)
22. Segundo, P.S.: A new DSATUR-based algorithm for exact vertex coloring. Comput. Oper. Res. **39**(7), 1724–1733 (2012)
23. Sewell, E.: An improved algorithm for exact graph coloring. In: Johnson and Trick [16], pp. 359–373
24. Trick, M.: DIMACS graph coloring instances (2002). http://mat.gsia.cmu.edu/COLOR02/

Submodular Maximization with Uncertain Knapsack Capacity

Yasushi Kawase[1(✉)], Hanna Sumita[2], and Takuro Fukunaga[3]

[1] Tokyo Institute of Technology, Tokyo, Japan
kawase.y.ab@m.titech.ac.jp
[2] JST, ERATO, Kawarabayashi Large Graph Project,
National Institute of Informatics, Tokyo, Japan
sumita@nii.ac.jp
[3] Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan
takuro.fukunaga@riken.jp

Abstract. We consider the maximization problem of monotone submodular functions under an uncertain knapsack constraint. Specifically, the problem is discussed in the situation that the knapsack capacity is not given explicitly and can be accessed only through an oracle that answers whether or not the current solution is feasible when an item is added to the solution. Assuming that cancellation of an item is allowed when it overflows the knapsack capacity, we discuss the robustness ratios of adaptive policies for this problem, which are the worst case ratios of the objective values achieved by the output solutions to the optimal objective values. We present a randomized policy of robustness ratio $(1-1/e)/2$, and a deterministic policy of robustness ratio $2(1-1/e)/21$. We also consider a universal policy that chooses items following a precomputed sequence. We present a randomized universal policy of robustness ratio $(1-1/\sqrt[4]{e})/2$. When the cancellation is not allowed, no randomized adaptive policy achieves a constant robustness ratio. Because of this hardness, we assume that a probability distribution of the knapsack capacity is given, and consider computing a sequence of items that maximizes the expected objective value. We present a polynomial-time randomized algorithm of approximation ratio $(1-1/\sqrt[4]{e})/4-\epsilon$ for any small constant $\epsilon > 0$.

1 Introduction

The submodular maximization is one of the most well-studied combinatorial optimization problems. Since it captures an essential part of decision-making situations, it has a huge number of applications in diverse areas of computer science. Nevertheless, the standard setting of the submodular maximization problem fails to capture several realistic situations. For example, let us consider choosing several items to maximize a reward represented by a submodular function subject to a resource limitation. When the amount of the available resource is exactly known, this problem is formulated as the submodular maximization problem with a knapsack constraint. However, in many practical cases, precise information on the available resource is not given. Thus, algorithms for the standard submodular maximization problem cannot be applied to this situation. Motivated by this fact, we study robust maximization of submodular functions with an uncertain knapsack capacity. Besides the practical applications, it is interesting to study this problem because it shows how much robustness can be achieved for an uncertain knapsack capacity in submodular maximization.

More specifically, we study the *submodular maximization problem with an unknown knapsack capacity* (SMPUC). In this problem, we are given a set I of items, and a monotone nonnegative submodular function $f\colon 2^I \to \mathbb{R}_+$ such that $f(\emptyset) = 0$, where each item $i \in I$ is associated with a size $s(i)$. The objective is to find a set of items that maximizes the submodular function subject to a knapsack constraint, but we assume that the knapsack capacity is unknown. We have access to the knapsack capacity through an oracle; we add items to the knapsack one by one, and we see whether or not the selected items violates the knapsack constraint only after the addition. If a selected item fits the knapsack, the selection of this item is irrevocable. When the total size of the selected items exceeds the capacity, there are two settings according to whether or not the selection can be canceled. If cancellation is allowed, then we remove the last selected item from the knapsack, and continue adding the remaining items to the knapsack. In the other setting, we stop the selection, and the final output is defined as the one before adding the last item.

For the setting where the cancellation is allowed, we consider an *adaptive policy*, which is defined as a decision tree to decide which item to pack into the knapsack next. Performance of an adaptive policy is evaluated by the robustness ratio defined as follows. For any number $C \in \mathbb{R}_+$, let OPT_C denote the optimal item set when the capacity is C, and let ALG_C denote an output of the policy. Note that if the policy is a randomized one, then ALG_C is a random variable. We call the adaptive policy *α-robust*, for some $\alpha \leq 1$, if for any $C \in \mathbb{R}_+$, the expected objective value of the policy's output is within a ratio α of the optimal value, i.e., $\mathbb{E}[f(\mathrm{ALG}_C)]/f(\mathrm{OPT}_C) \geq \alpha$. We also call the ratio α the *robustness ratio* of the policy.

One main purpose of this paper is to present algorithms that produce adaptive policies of constant robustness ratios for SMPUC. Moreover, we discuss a special type of adaptive policy called a *universal* policy. Such a policy selects items following a precomputed order of items regardless of the observations made while packing. This is in contrast to general adaptive policies, where the next item to try can vary with the observations made up to that point. We present an algorithm that produces a randomized universal policy that achieves a constant robustness ratio.

If cancellation is not allowed, then there is no difference between adaptive and universal policies because the selection terminates once a selected item does not fit the knapsack. In this case, we observe that no randomized adaptive policy achieves a constant robustness ratio. Due to this hardness, we consider a *stochastic* knapsack capacity when cancellation is not allowed. In this situation, we assume that the knapsack capacity is determined according to some probability distribution and the information of the distribution is available. Based on this assumption, we compute a sequence of the given items as a solution. When the knapsack capacity is realized, the items in the prefix of the sequence are selected so that their total size does not exceed the realized capacity. The objective of the problem is to maximize the expected value of the submodular function f for the selected items. We address this problem as the *submodular maximization problem with a stochastic knapsack capacity* (SMPSC). We say that the *approximation ratio* of a sequence is α (≤ 1) if its expected objective value is at least α times the maximum expected value of f for any instance. The sequence computed in an α-robust policy for SMPUC achieves α-approximation ratio for SMPSC. However, the opposite does not hold, and an algorithm of a constant approximation ratio may exist for SMPSC even though no randomized adaptive policy achieves a constant robustness ratio. Indeed, we present such an algorithm.

Related studies: There are a huge number of studies on the submodular maximization problems (e.g., [18]), but we are aware of no previous work on SMPUC or SMPSC. Regarding studies on the stochastic setting of the problem, several papers proposed concepts of submodularity for random set functions and discussed adaptive policies to maximize those functions [2,10]. There are also studies on the submodular maximization over an item set in which each item is activated stochastically [1,9,12]. However, as far as we know, there is no study on the problem with stochastic constraints.

When the objective function is modular (i.e., the function returns the sum of the weights associated with the selected items), the submodular maximization problem with a knapsack constraint is equivalent to the classic knapsack problem. For the knapsack problem, there are numerous studies on the stochastic sizes and rewards of items [5,11,15]. This problem is called the stochastic knapsack problem. Note that this is different from the knapsack problem with a stochastic capacity (KPSC), and there is no direct relationship between them.

The covering version of KPSC is studied in a context of single machine scheduling with nonuniform processing speed. This problem is known to be strongly NP-hard [13], which implies that pseudo-polynomial time algorithms are unlikely to exist. This is in contrast to the fact that the classic knapsack problem and its covering version admit pseudo-polynomial time algorithms. Megow and Verschae [17] gave a PTAS for the covering version of KPSC.

To the best of our knowledge, KPSC itself has not been studied well. The only previous study we are aware of is the thesis of Dabney [4], wherein a PTAS is presented for the problem. Since the knapsack problem and its covering version are equivalent in the existence of exact algorithms, the strongly NP-hardness of the covering version implies the same hardness for the knapsack problem.

Regarding the *knapsack problem with an unknown capacity* (KPUC), Megow and Mestre [16] mentioned that no deterministic policy achieves a constant robustness ratio when cancellation is not allowed. They presented an algorithm that constructs for each instance a policy whose robustness ratio is arbitrarily close to the one of an optimal policy that achieves the largest robustness ratio for the instance. When cancellation is allowed, Disser et al. [6] provided a deterministic 1/2-robust universal policy for KPUC. They also proved that no deterministic adaptive policy achieves a robustness ratio better than 1/2, which means that the robustness ratio of their deterministic universal policy is best possible even for any deterministic adaptive policy.

Contributions: For the case where cancellation is allowed, we present three polynomial-time algorithms for SMPUC. These algorithms produce

– a randomized adaptive policy of robustness ratio $(1 - 1/e)/2$ (Sect. 3);
– a deterministic adaptive policy of robustness ratio $2(1 - 1/e)/21$ (Sect. 4);
– a randomized universal policy of robustness ratio $(1 - 1/\sqrt[4]{e})/2$ (Sect. 5).

Our algorithms are based on a simple greedy algorithm [14] for the monotone submodular maximization problem with a knapsack constraint. The greedy algorithm outputs a better solution among two candidates, one of which is constructed greedily based on the increase in the objective function value per unit of size, and the other of which is based on the increase in the objective function value. In our randomized adaptive policy, we achieve the robustness ratio $(1 - 1/e)/2$ by guaranteeing that each of the two candidate solutions is output by our policy with probability 1/2.

We convert this randomized policy into a deterministic one by mixing the two strategies which correspond to the two candidate solutions. We remark that the same approach is taken for KPUC to construct a deterministic 1/2-robust universal policy by Disser et al. [6]. They call an item *swap item* if it corresponds to a single-item solution for some knapsack capacity. A key idea in their policy is to pack swap items earlier than the others. However, their technique fully relies on the property that the objective function is modular, and their choice of swap items is not suitable for SMPUC. In this paper, we introduce a new notion of *single-valuable items*. This enables us to design a deterministic $2(1 - 1/e)/21$-robust policy. We remark that our proof technique is also different from the standard one used in related work. Moreover, we modify the randomized adaptive policy to obtain the randomized universal policy. A key idea here is to guess a capacity by a doubling strategy.

We also show that no randomized adaptive policy achieves a robustness ratio better than 8/9 for KPUC. It is known that the robustness ratio achieved by deterministic policies for this problem is at most 1/2 [6], but there was no upper bound on the robustness ratio for randomized policies.

When cancellation is not allowed, it has been already known that KPUC admits no deterministic universal policy of a constant robustness ratio [16]. We can observe that this hardness result by showing that no randomized adaptive policy achieves a constant robustness ratio.

Because of this hardness, we consider SMPSC without cancellation. We present a polynomial-time randomized algorithm of approximation ratio $(1 - 1/\sqrt[4]{e})/4 - \epsilon$ for any small constant $\epsilon > 0$ (Sect. 6). This algorithm is based on the idea of Gupta et al. [11] for the stochastic knapsack problem.

Gupta et al. regarded the knapsack capacity as a time limit, and showed that a rounding algorithm for a time-index linear program (LP) gives an adaptive policy for the stochastic knapsack problem. Although the formulation size of the time-index LP is not polynomial, a simple doubling technique reduces the formulation size to polynomial with a loss of the approximation ratio. In our algorithm for SMPSC, we first introduce a time-index convex relaxation of the problem using the multilinear extension of the objective function, and show that the rounding algorithm of Gupta et al. is a monotone contention resolution scheme for any realization of the knapsack capacity. This observation gives a pseudo-polynomial time $(1 - 1/\sqrt[4]{e})/2$-approximation algorithm for SMPSC. We then transform it into a polynomial-time algorithm. This transformation requires a careful sketching of knapsack capacity, which was not necessary for the stochastic knapsack problem.

Organization: The rest of this paper is organized as follows. Section 2 gives notations and preliminary facts used in this paper. Sections 3, 4, and 5 present the adaptive policies for SMPUC with cancellation. Section 6 presents the polynomial-time approximation algorithm for SMPSC without cancellation. Due to the space limitation, we omit several proofs and supplementary results including the above-mentioned upper-bound and hardness results; see the upcoming full version for these.

2 Preliminaries

In this section, we define terminologies used in this paper, and introduce existing results which we will use.

Maximization of monotone submodular functions: The inputs of the problem are a set I of n items and a nonnegative set function $f : 2^I \to \mathbb{R}_+$. In this paper, we assume that (i) f satisfies $f(\emptyset) = 0$, (ii) f is *submodular* (i.e., $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ for any $X, Y \subseteq I$), and (iii) f is *monotone* (i.e., $f(X) \leq f(Y)$ for any $X, Y \subseteq I$ with $X \subseteq Y$). Function f is given as an oracle that returns the value of $f(X)$ for any query $X \subseteq I$. Let $\mathcal{I} \subseteq 2^I$ be any family such that $X \subseteq Y \in \mathcal{I}$ implies $X \in \mathcal{I}$. The $\mathcal{I}$-constrained submodular maximization problem seeks finding $X \in \mathcal{I}$ that maximizes $f(X)$.

We focus on the case where $\mathcal{I}$ corresponds to a knapsack constraint. Namely, each item $i \in I$ is associated with a size $s(i)$, and $\mathcal{I}$ is defined as $\{X \subseteq I\colon \sum_{i \in X} s(i) \leq C\}$ for some knapsack capacity $C > 0$. We assume that the item size $s(i)$ $(i \in I)$ and the knapsack capacity C are positive integers. We denote $\sum_{i \in X} s(i)$ by $s(X)$ for any $X \subseteq I$.

Problem SMPUC: In SMPUC, the knapsack capacity C is unknown. We see whether items in the knapsack fits the knapsack only when adding an item to the knapsack.

A solution for SMPUC is an adaptive policy $\mathcal{P}$, which is represented as a binary decision tree that contains every item at most once along each path from the root to a leaf. Each node of the decision tree is an item to try packing into the knapsack. A *randomized* policy is a probability distribution over binary decision trees. One of the decision trees is selected according to the probability distribution. For a fixed capacity C, the output of a policy $\mathcal{P}$ is an item set denoted by $\mathcal{P}(C) \subseteq I$ obtained as follows. We start with $\mathcal{P}(C) = \emptyset$ and check whether the item r at the root of $\mathcal{P}$ fits the knapsack, i.e., whether $s(r) + s(\mathcal{P}(C)) \leq C$. If the item fits, then we add r to $\mathcal{P}(C)$ and continue packing recursively with the left subtree of r. Otherwise, we have two options: when cancellation is allowed, we discard r and continue packing recursively with the right subtree of r; when cancellation is not allowed, we discard r and output $\mathcal{P}(C)$ to terminate the process.

When a policy does not depend on the observation made while packing, we call such a policy universal. Since every path from the root to a leaf in a universal policy is identical, we can identify a universal policy with a sequence $\Pi = (\Pi_1, \ldots, \Pi_n)$ of items in I. For a fixed capacity C, the output of a universal policy, denoted by $\Pi(C)$, is constructed as follows. We start with $\Pi(C) = \emptyset$. For each $i = 1, \ldots, n$, we check whether $s(\Pi(C)) + s(\Pi_i) \leq C$ holds or not. If true, then Π_i is added to X. Otherwise, Π_i is discarded, and we proceed to the next i when cancellation is allowed, and we terminate the process when cancellation is not allowed.

Problem SMPSC: In SMPSC, the knapsack capacity C is given according to some probability distribution. Let $T = \sum_{i \in I} s(i)$. For each $t \in [T] := \{0, 1, \ldots, T\}$, we denote by $p(t)$ the probability that the knapsack capacity is t. We assume that the probability is given to an algorithm through an oracle that returns the value of $\sum_{t'=t}^{T} p(t')$ for any query $t \in [T]$. Hence, the input size of a problem instance is $O(n \log T)$ and an algorithm runs in pseudo-polynomial time if its running time depends on T linearly. A solution for SMPSC is a universal policy, i.e., a sequence $\Pi = (\Pi_1, \ldots, \Pi_n)$ of the items in I. When a capacity C is decided, the output $\Pi(C)$ of Π is constructed in the same way as universal policies for SMPUC. The objective of SMPSC is to find a sequence Π that maximizes $\mathbb{E}[f(\Pi(C))]$.

Multilinear extension, continuous greedy, and contention resolution scheme: From any vector $x \in [0, 1]^I$, we define a random subset R_x of I so that each $i \in I$ is included in R_x with probability x_i, where the inclusion of i is independent from the inclusion of the other items. For a submodular function $f\colon 2^I \to \mathbb{R}_+$, its *multilinear extension* $F\colon [0, 1]^I \to \mathbb{R}_+$ is defined by $F(x) = \mathbb{E}[f(R_x)] = \sum_{X \subseteq I} f(X) \prod_{i \in X} x_i \prod_{i' \in I \setminus X}(1 - x_{i'})$ for all $x \in [0, 1]^I$. This function F satisfies the *smooth monotone submodularity*, that is, $\frac{\partial F(x)}{\partial x_i} \geq 0$ for any $i \in I$ and $\frac{\partial^2 F(x)}{\partial x_i \partial x_j} \geq 0$ for any $i, j \in I$.

A popular approach for solving the $\mathcal{I}$-constrained submodular maximization problem is to use a continuous relaxation of the problem. Let $P \subseteq [0, 1]^I$ be a polytope in which each integer solution is the incidence vector of a member of $\mathcal{I}$. Then, $\max_{x \in P} F(x) \geq \max_{X \in \mathcal{I}} f(X)$ holds. In this approach, it is usually assumed that P is *downward-closed* (i.e., if $x, y \in [0, 1]^I$ satisfies $y \leq x \in P$, then $y \in P$), and *solvable* (i.e., the maximization problem $\max_{x \in P} \sum_{i \in I} w_i x_i$ can be solved in polynomial time for any $w \in \mathbb{R}_+^I$).

Calinescu et al. [3] gave an algorithm called *continuous greedy* for a continuous maximization problem $\max_{x \in P} F(x)$ over a solvable downward-closed polytope P. They proved that the continuous greedy outputs a solution $x \in P$ such that $F(x) \geq (1 - 1/e - o(1)) \max_{x' \in P} F(x')$. Feldman [7] extended its analysis by observing that the continuous greedy algorithm with stopping time $b \geq 0$ outputs a solution $x \in [0,1]^I$ such that $x/b \in P$ and $F(x) \geq (1 - e^{-b} - o(1)) \max_{X \in \mathcal{I}} f(X)$. (The performance guarantee depending on the stopping time is originally given for the measured continuous greedy algorithm proposed by [8].) It is easy to see that his analysis can be modified to prove a slightly stronger result $F(x) \geq (1 - e^{-b} - o(1)) \max_{x' \in P} F(x')$. In addition, this analysis requires only the smooth monotone submodularity as a property of F.

A fractional solution $x \in P$ can be rounded into an integer solution by a contention resolution scheme. Let $b, c \in [0,1]$. For a vector x, we denote $\mathrm{supp}(x) = \{i \in I : x_i > 0\}$. We consider an algorithm that receives $x \in bP$ and $A \subseteq I$ as input and returns a random subset $\pi_x(A) \subseteq A \cap \mathrm{supp}(x)$. Such an algorithm is called (b,c)- *balanced contention resolution scheme* if $\pi_x(A) \in \mathcal{I}$ in probability 1 for all x and A, and $\Pr[i \in \pi_x(R_x) : i \in R_x] \geq c$ holds for all x and $i \in \mathrm{supp}(x)$ (recall that R_x is the random subset of I determined from x). It is also called *monotone* if $\Pr[i \in \pi_x(A)] \geq \Pr[i \in \pi_x(A')]$ for any $i \in A \subseteq A' \subseteq I$. If a monotone (b,c)-balanced contention resolution scheme is available, then we can achieve the approximation ratio claimed in the following theorem by applying it to a fractional solution computed by the measured continuous greedy algorithm with stopping time b. This fact is summarized as in the following theorem.

Theorem 1 **([8]).** *If there exists a monotone (b,c)-balanced contention resolution scheme for $\mathcal{I}$, then the $\mathcal{I}$-constrained submodular maximization problem admits an approximation algorithm of ratio $(1 - e^{-b})c - o(1)$ for any nonnegative monotone submodular function.*

3 Randomized $(1 - 1/e)/2$-Robust Adaptive Policy for SMPUC

In this section, we present a randomized adaptive policy for SMPUC in the situation that cancellation is allowed. The idea of our algorithm is based on a simple greedy algorithm [14] for the submodular maximization problem with a knapsack constraint. The greedy algorithm generates two candidate item sets. One set is obtained greedily by repeatedly inserting an item maximizing the increase in the objective function value per unit of size. The other is obtained similarly by packing an item maximizing the increase in the objective function value. Then the algorithm returns the set with the larger total value, which leads to a $(1 - 1/e)/2$-approximation solution.

The idea of choosing a better solution is not suitable for SMPUC, where we cannot remove items from the knapsack. We resolve this issue by generating at random one of two policies $\mathcal{P}^1$ and $\mathcal{P}^2$ which are analogous to the above two greedy methods. One policy $\mathcal{P}^1$ corresponds to the greedy algorithm based on

Algorithm 1. Greedy algorithm $\mathcal{P}^1$ for (I, U)

```
1  $X \leftarrow U$, $R \leftarrow I \setminus U$;
2  foreach $j = 1, \dots, |I \setminus U|$ do
3      let $i_j \in \arg\max\{(f(X \cup \{i\}) - f(X))/s(i) \ : \ i \in R\}$;
4      if $i_j$ fits the knapsack (i.e., $s(X) + s(i_j) \leq C$) then // left subtree
5          $X \leftarrow X \cup \{i_j\}$
6      else // right subtree
7          discard $i_j$
8      $R \leftarrow R \setminus \{i_j\}$;
```

Algorithm 2. Randomized $(1 - 1/e)/2$-robust adaptive policy

```
1  flip a coin;
2  if head then execute Algorithm 1 for $(I, \emptyset)$;
3  else
4      $X \leftarrow \emptyset$, $R \leftarrow I$;
5      foreach $j = 1, \dots, |I|$ do
6          let $i_j \in \arg\max\{f(X \cup \{i\}) - f(X) \ : \ i \in R\}$;
7          if $i_j$ fits the knapsack (i.e., $s(X) + s(i_j) \leq C$) then // left subtree
8              $X \leftarrow X \cup \{i_j\}$
9          else // right subtree
10             discard $i_j$
11         $R \leftarrow R \setminus \{i_j\}$;
```

the increase in the objective function value per unit of size. We formally present this policy as Algorithm 1. The item i_j corresponds to a node of depth $j-1$ in $\mathcal{P}^1$. We remark that Algorithm 1 chooses i_j independently of the knapsack capacity, but the choice depends on the observations which items fit the knapsack and which items did not fit so far. For generality of the algorithm, we assume that the algorithm receives an initial state U of the knapsack, which is defined as a subset of I (this will be used in Sect. 4).

The policy $\mathcal{P}^2$ tries to pack items based on the increase in the objective function value. Our algorithm is summarized in Algorithm 2. We remark that Algorithm 2 chooses the item i_j for each iteration j in polynomial time with respect to the cardinality of I.

We analyze the robustness ratio of Algorithm 2. In execution of Algorithm 1 for (I, U) under some capacity, we call the order $(i_1, \dots, i_{|I \setminus U|})$ of items in $I \setminus U$ the *greedy order* for (I, U), where i_j is the jth selected item at line 3. Note that the greedy order depends on the capacity. A key concept in the analysis of Algorithm 2 is to focus on the first item in the greedy order that is a member of OPT_C but is spilled from $\mathcal{P}^1(C)$. The following lemma is useful in analysis of Algorithm 2 and also algorithms given in subsequent sections.

Lemma 1. *Let C, C' be any positive numbers. Let q be the smallest index such that $i_q \in \mathrm{OPT}_{C'}$ and $i_q \notin \mathcal{P}^1(C)$ (let $q = \infty$ if there is no such index). When Algorithm 2 is executed for (I, U) with capacity C, it holds for any index j that*

$$\begin{aligned} & f(((\mathcal{P}^1(C) \cup \mathrm{OPT}_{C'}) \cap \{i_1, \ldots, i_j\}) \cup U) \\ & \geq \left(1 - \exp\left(-s((\mathcal{P}^1(C) \cup \mathrm{OPT}_{C'}) \cap \{i_1, \ldots, i_j\})/C'\right)\right) \cdot f(\mathrm{OPT}_{C'}). \end{aligned}$$

Moreover, $(\mathcal{P}^1(C) \cup \mathrm{OPT}_{C'}) \cap \{i_1, \ldots, i_j\} = \mathcal{P}^1(C) \cap \{i_1, \ldots, i_j\}$ holds for any $j < q$.

Theorem 2. *Algorithm 2 is a randomized $(1 - 1/e)/2 > 0.316$-robust adaptive policy.*

4 Deterministic $2(1 - 1/e)/21$-Robust Adaptive Policy for SMPUC

In this section, we present a deterministic adaptive policy for SMPUC by modifying Algorithm 2. To this end, let us review the result of Disser et al. [6] for KPUC, which is identical to SMPUC with modular objective functions. They obtained a deterministic 1/2-robust universal policy for KPUC based on the greedy order for $(I, \emptyset)$. Their policy first tries to insert items with large values, which are called *swap items*. For a greedy order $(i_1, \ldots, i_n)$, an item i_j is called a swap item if $f(\{i_j\}) \geq f(\mathcal{P}^1(C))$ for some capacity C such that i_j is the first item that overflows the knapsack when the items are packed in the greedy order. The key property is that the greedy order does not depend on the capacity C when f is modular. This enables them to determine swap items from the unique greedy order, and to obtain a deterministic universal policy.

On the other hand, it is hard to apply their idea to our purpose. The difficulty is that the greedy order depends on the capacity when f is submodular. Thus the notion of swap items is not suitable for choosing the items that should be tried first. In this paper, we introduce *single-valuable items*, which are items i satisfying $f(\{i\}) \geq 2 \cdot f(\mathrm{OPT}_{s(i)/2})$ $(= 2 \cdot \max\{f(X) : s(X) \leq s(i)/2\})$. In the design of our algorithm, we use a polynomial-time γ-approximation algorithm that computes $f(\mathrm{OPT}_{s(i)/2})$; for example, $\gamma = 1 - 1/e$ [18]. Our algorithm calculates the set S of the single-valuable items in a sense of γ-approximation. To be precise, it holds that $f(\{i\}) \geq 2\gamma \cdot f(\mathrm{OPT}_{s(i)/2})$ for any item $i \in S$ and $f(\{i\}) \leq 2 \cdot f(\mathrm{OPT}_{s(i)/2})$ for any item $i \notin S$.

Our algorithm tries to insert items in S first until one of these items fits the knapsack (or all the items have been canceled) and then it executes Algorithm 1 with the remaining items. We remark that, unlike the algorithm for KPUC, our algorithm executes Algorithm 1 once a single-valuable item fits the knapsack. We summarize our algorithm in Algorithm 3.

Note that our algorithm constructs S and decides which item to try in polynomial time. Let R and U be the sets at the beginning of line 10. Then U is empty if $S \cap I_C = \emptyset$, and U consists of exactly one item $i^* \in \arg\max\{f(\{i\}) : i \in I_C \cap S\}$ otherwise. The following theorem is the main result of this section.

Algorithm 3. Deterministic $2\gamma/21$-robust policy

```
1  $U \leftarrow \emptyset$, $R \leftarrow I$, $S \leftarrow \emptyset$;
2  foreach $i \in I$ do
3      Let $L$ be a $\gamma$-approximate solution to $\max\{f(X) : s(X) \leq s(i)/2,\ X \subseteq I\}$;
4      if $f(\{i\}) \geq 2f(L)$ then $S \leftarrow S \cup \{i\}$;
5  while $U = \emptyset$ and $S \cap R \neq \emptyset$ do
6      let $i \in \arg\max\{f(\{i\}) : i \in S \cap R\}$;
7      if $i$ fits the knapsack (i.e., $s(i) \leq C$) then $U \leftarrow \{i\}$;// left subtree
8      else discard $i$;// right subtree
9      $R \leftarrow R \setminus \{i\}$;
10 execute Algorithm 1 for $(R \cup U, U)$;
```

Theorem 3. *Algorithm 3 using a γ-approximation algorithm in line 3 is a $\min\{2\gamma/21, (1 - 1/\sqrt[3]{e})/3\}$-robust universal policy. In particular, it is $2(1 - 1/e)/21 > 0.060$-robust when $\gamma = 1 - 1/e$, and it is $(1 - 1/\sqrt[3]{e})/3 > 0.094$-robust when $\gamma = 1$.*

We describe the proof idea. Suppose that the given capacity is C. Since the output of Algorithm 3 is $\mathcal{P}^1(C)$ for $(R \cup U, U)$, one can think of a similar proof to the one of Theorem 2. However, we may not be able to use the value $f(\{i_q\})$ in the evaluation of $f(\mathcal{P}^1(C))$ here, because $\mathcal{P}^1(C)$ may not contain some items that bound $f(\{i_q\})$. We show the theorem using a different approach. A basic idea is to divide OPT_C into several subsets $A_1, \ldots, A_k$ and derive a bound $f(\mathrm{OPT}_C) \leq f(A_1) + \cdots + f(A_k)$ by the submodularity of f. A key idea is to evaluate each $f(A_i)$ using properties of single-valuable items, which are shown as the following two lemmas.

Lemma 2. *We have $f(\{i\}) \leq \max\{f(U), 2f(\mathrm{OPT}_{s(i)/2})\}$ for any item $i \in I_C$.*

Proof. The lemma follows because $f(\{i\}) \leq f(U)$ if $i \in S$ and $f(\{i\}) \leq 2f(\mathrm{OPT}_{s(i)/2})$ if $i \notin S$. □

Lemma 3. *Let C be any positive number. Denote $s^* = s(U)$. Then for any number $x \in [s^*, C/2]$, it holds that $f(\mathrm{OPT}_{2x}) \leq 3 \cdot f(\mathrm{OPT}_x)$.*

Proof of Theorem 3. Let $\mathcal{P}$ be the deterministic policy described as Algorithm 3. Suppose that the given capacity is C. We remark that $\mathcal{P}(C) = \mathcal{P}^1(C)$ for $(R \cup U, U)$. We may assume that $\mathrm{OPT}_C \not\subseteq \mathcal{P}(C)$ since otherwise $f(\mathcal{P}(C)) = f(\mathrm{OPT}_C)$. Let $s^* = s(U)$. Let i^* denote the unique item of U when $I_C \cap S \neq \emptyset$. We branch the analysis into two cases: (a) $s^* < C/3$ and (b) $s^* \geq C/3$.

Case (a): We claim that $f(\mathcal{P}(C)) \geq (1 - 1/\sqrt[3]{e}) \cdot f(\mathrm{OPT}_C)/3$. Since $s^* < C/3 < C/2$, Lemma 3 indicates that $f(\mathrm{OPT}_{C/2}) \geq f(\mathrm{OPT}_C)/3$. We evaluate $f(\mathrm{OPT}_{C/2})$ by using Lemma 1 with $C' = C/2$. We may assume that $\mathrm{OPT}_{C/2} \not\subseteq \mathcal{P}(C)$; otherwise $f(\mathcal{P}(C)) \geq f(\mathrm{OPT}_{C/2})$ holds, which implies that $f(\mathcal{P}(C)) \geq f(\mathrm{OPT}_C)/3$. Let $(i_1, \ldots, i_{|R \cup U|})$ be the greedy order for $(R \cup U, U)$. Let q' be

Algorithm 4. Randomized $(1 - 1/\sqrt[4]{e})/2$-robust universal policy

```
1 $\Pi \leftarrow ()$;
2 flip a coin;
3 if head then
4     $l \leftarrow 1$, $s_{\min} \leftarrow \min_{i \in I} s(i)$;
5     for $k \leftarrow 0$ to $\lceil \log_2(\sum_{i \in I} s(i)/s_{\min}) \rceil$ do
6         let $Y^{(k)}$ be the output $\mathcal{P}^1(2^k \cdot s_{\min})$ of the policy given in Algorithm 1 for $(I, \emptyset)$;
7         foreach $i \in Y^{(k)} \setminus \bigcup_{j=1}^{k-1} Y^{(j)}$ do $\Pi_l \leftarrow i$, $l \leftarrow l + 1$;
8 else let $\Pi$ be the decreasing order of items $i \in I$ in value $f(\{i\})$ ;
```

the smallest index such that $i_{q'} \in \mathrm{OPT}_{C/2}$ and $i_{q'} \notin \mathcal{P}^1(C)$. We denote $Z = \mathcal{P}^1(C) \cap \{i_1, \ldots, i_{q'-1}\}$. As $s(i_{q'}) \leq C/2$, we see that $s(Z) > C - s^* - s(i_{q'}) \geq C/6$. Then Lemma 1 implies that $f(\mathcal{P}(C)) \geq f(Z \cup U) \geq (1 - 1/\sqrt[3]{e}) \cdot f(\mathrm{OPT}_{C/2})$, and hence the claim follows.

Case (b): By the following claim, we obtain $f(\mathcal{P}(C)) \geq (2\gamma/21) \cdot f(\mathrm{OPT}_C)$.

Claim. It holds that $f(\mathrm{OPT}_C) \leq 7 f(\mathrm{OPT}_{s^*})$ and $f(\mathrm{OPT}_{s^*}) \leq \frac{3}{2\gamma} \cdot f(\{i^*\})$. □

5 Randomized $(1 - 1/\sqrt[4]{e})/2$-Robust Universal Policy for SMPUC

In this section, we devise a randomized $(1 - 1/\sqrt[4]{e})/2$-robust universal policy by modifying Algorithm 2. Note that we cannot use directly Algorithm 1 in universal policies, because the greedy order depends on the capacity. Instead we guess the capacity by the doubling strategy and emulate the execution of Algorithm 1. In each iteration of our algorithm, it finds an item set X maximizing $f(X)$ under a bound C' on the total size $s(X)$, and then appends items of X to the sequence. The bound C' is set by the algorithm according to the input items. To compute the set X, we use Algorithm 1 where the capacity is set to be C'. We double the bound after each iteration. Our algorithm is summarized in Algorithm 4.

We remark that Algorithm 4 constructs a sequence of items in polynomial time with respect to the input size. We can prove the following result by using Lemma 1.

Theorem 4. *Algorithm 4 is a* $(1 - 1/\sqrt[4]{e})/2 > 0.110$*-robust randomized universal policy.*

6 Polynomial-Time $((1 - 1/\sqrt[4]{e})/4 - \epsilon)$-Approximation Algorithm for SMPSC

In this section, we consider SMPSC in the situation where cancellation is not allowed. We present a polynomial-time algorithm of approximation ratio

$(1-1/\sqrt[4]{e})/4-\epsilon$ for any small constant $\epsilon>0$. This algorithm is based on the idea of Gupta et al. [11] for the stochastic knapsack problem. We first give a pseudo-polynomial time $(1-1/\sqrt[4]{e})/2$-approximation algorithm, and then we transform it into a polynomial-time algorithm.

Our algorithm relies on a continuous relaxation of the problem. The relaxation is formulated based on an idea of using time-indexed variables; we regard the knapsack capacity as a time limit while considering that picking an item i spends time $s(i)$. In the relaxation, we have a variable $x_{ti}\in[0,1]$ for each $t\in[T-1]$ and $i\in I$, and $x_{ti}=1$ represents that item i is picked at time t. Recall that $T=\sum_{i\in I}s(i)$ and $[T']=\{0,1,\dots,T'\}$. For each $t\in[T]$ and $i\in I$, let $\bar{x}_{ti}=\sum_{t'\in[t-s(i)]}x_{t'i}$, where we define the right-hand side as 0 if $t<s(i)$. For each $t\in[T]$, let $\bar{x}_t$ be the $|I|$-dimensional vector whose component corresponding to $i\in I$ is $\bar{x}_{ti}$. Let $F\colon[0,1]^I\to\mathbb{R}_+$ be the multilinear extension of the submodular function f. Then, the relaxation is described as

$$\begin{array}{lll}\text{maximize} & \bar{F}(x):=\sum_{t=1}^{T}p(t)F(\bar{x}_t) & \\ \text{subject to} & \sum_{t\in[T-1]}x_{ti}\le 1, & \forall i\in I,\\ & \sum_{i\in I}\sum_{t'\in[t]}x_{t'i}\min\{s(i),t\}\le 2t, & \forall t=1,\dots,T,\\ & x_{ti}\ge 0, & \forall t\in[T-1],\forall i\in I.\end{array} \tag{1}$$

Let us see that (1) relaxes the problem. It is not difficult to see that the first and the third constants are valid. We prove that the second constraint is valid. Suppose that x is an integer solution that corresponds to a sequence of items. Let I_t be the set of items picked at time t or earlier in this solution. Notice that $\sum_{i\in I}\sum_{t'\in[t]}x_{t'i}\min\{s(i),t\}=\sum_{i\in I_t}\min\{s(i),t\}$ holds. Let j be the item picked latest in I_t. Then, since the process of all items in $I_t\setminus\{j\}$ terminates by time t, we have $\sum_{i\in I_t\setminus\{j\}}s(i)\le t$. Therefore, $\sum_{i\in I_t}\min\{s(i),t\}=\min\{s(j),t\}+\sum_{i\in I_t\setminus\{j\}}s(i)\le 2t$.

Note also that $\bar{F}$ is a smooth monotone submodular function; i.e., $\partial\bar{F}(x)/\partial x_{ti}\ge 0$ for any $t\in[T-1]$ and $i\in I$, and $\frac{\partial^2\bar{F}(x)}{\partial x_{ti}\partial x_{t'i'}}\le 0$ for any $t,t'\in[T-1]$ and $i,i'\in I$. Hence, we can apply the continuous greedy algorithm for solving (1). Let x^* be an obtained feasible solution for (1). We first present a rounding algorithm for this solution. Since the formulation size of this relaxation is not polynomial, this algorithm does not run in polynomial time. We convert the algorithm into a polynomial-time one later.

The algorithm consists of two rounds. In the first round, each item i chooses an integer t from $[T-1]$ with probability $x^*_{ti}/4$, and chooses no integer with probability $1-\sum_{t\in[T-1]}x^*_{ti}/4$. An item is discarded if it chooses no integer. Let I_1 be the set of remaining items. For each $i\in I_1$, let t_i denote the integer chosen by i.

Then, the algorithm proceeds to the second round. Let $i\in I_1$, and let J_i denote $\{j\in I_1\colon t_j\le t_i\}$. In the second round, item i is discarded if $\sum_{j\in J_i}s(j)\ge t_i$. Let I_2 denote the set of items remaining after the second round. The algorithm outputs a sequence obtained by sorting the items $i\in I_2$ in the non-decreasing order of t_i, where ties are broken arbitrarily while the other items follow those in I_2 in an arbitrary order.

For $t \in [T]$, let $I_t = \{i \in I_2 \colon t_i \leq t - s(i) - 1\}$. If $i \in I_t$, then i contributes to the objective value of the solution when the knapsack capacity is at least t.

Lemma 4. *For any $t \in [T]$, I_t is the output of a monotone $(1/4, 1/2)$-balanced contention resolution scheme for the maximization problem of f under the knapsack capacity t and the fractional solution $\bar{x}^*_t$. Hence, the sequence output by the algorithm achieves an objective value of at least $\bar{F}(x^*/4)/2$ in expectation.*

By Theorem 1 and Lemma 4, our algorithm achieves $(1 - 1/\sqrt[4]{e})/2$-approximation if it is combined with the continuous greedy algorithm with stopping time $1/4$.

Lemma 4 also implies that the integrality gap of (1) is at least $1/8$. On the other hand, there exist some instances indicating that the integrality gap is at most $1/3 + \epsilon$ for any $\epsilon > 0$ even when the objective function is modular; see the upcoming full version for its detail.

Conversion into a polynomial-time algorithm: Let $W = f(I)$ and $w = \min_{i \in I} f(\{i\})$. We assume $w > 0$ without loss of generality; if $f(\{i\}) = 0$ for an item $i \in I$, we can safely remove i from I because the submodularity implies $f(S) = 0$ for any $S \subseteq I$ with $i \in S$. We assume that the input size of an instance is $\Omega(\log(W/w))$, and hence we say that an algorithm runs in polynomial time if its running time is expressed as a polynomial on $\log(W/w)$. Let ϵ be a positive constant smaller than 1.

For $t \in [T-1]$, we define $\bar{p}(t)$ as $\sum_{t'=t+1}^{T} p(t')$. Let $\eta = \lfloor \log_{1-\epsilon}(\epsilon w/(W \log T)) \rfloor$. For $j \in \{1, \ldots, \eta + 1\}$, let τ'_j be the minimum integer $t \in [T-1]$ such that $\bar{p}(t) < (1-\epsilon)^{j-1}$. We assume without loss of generality that $\bar{p}(\min_{i \in I} s(i)) = 1$, which means $\tau'_1 \geq \min_{i \in I} s(i)$. We denote the set of positive integers in $\{\tau'_j \colon j = 1, \ldots, \eta + 1\} \cup \{2^j \colon j \in [\lceil \log T \rceil - 1]\}$ by $\{\tau_1, \ldots, \tau_q\}$, and assume without loss of generality that $1 = \tau_1 < \tau_2 < \cdots < \tau_q$. We also let $\tau_0 = 0$ and $\tau_{q+1} = T$. We also define q_η so that $\tau_{q_\eta} = \tau'_{\eta+1}$. Because of the definition, integers $\tau_0, \ldots, \tau_{q+1}$ satisfy the following conditions:

- $\bar{p}(\tau_j) \geq \bar{p}(\tau_{j+1} - 1) \geq (1-\epsilon)\bar{p}(\tau_j)$ holds for any $j \in [q_\eta]$;
- $\bar{p}(\tau_j) < \epsilon w/(W \log T)$ for any $j \in \{q_\eta + 1, \ldots, q\}$;
- $\tau_j < \tau_{j+1} \leq 2\tau_j$ holds for any $j = 1, \ldots, q$.

Moreover, $q = O(\log T + \log(W/(w\epsilon)))$.

In addition, we define the set of integers in $\{\tau_0, \ldots, \tau_{q+1}\} \cup \{\tau_j - s(i) + 1 \colon j \in \{1, \ldots, q+1\}, i \in I\}$ as $\{\xi_0, \ldots, \xi_{r+1}\}$, where $0 = \xi_0 < \xi_1 < \ldots < \xi_r < \xi_{r+1} = T$. Notice that $r = O(n \log T + n \log(W/(w\epsilon)))$.

Instead of (1), our polynomial-time algorithm is based on a more compact relaxation, which has a variable y_{ki} for each $k \in [r]$ and $i \in I$. Roughly speaking, y_{ki} corresponds to variables $x_{\xi_k, i}, \ldots, x_{\xi_{k+1}-1, i}$ in (1). For $j = 1, \ldots, q+1$ and $i \in I$, we define an auxiliary variable z_{ji} as $\sum_{\xi_{k+1}-1 \leq \tau_j - s(i)} y_{ki}$, and define z_j as

Algorithm 5. Randomized $(1-\epsilon)(1-\epsilon-1/\sqrt[4]{e})/4$-approximation algorithm

```
for $\forall j \in \{1, \dots, \eta+1\}$ do
    compute $\tau'_j = \arg\min\{t \in [T-1] \colon \bar{p}(t) < (1-\epsilon)^{j-1}\}$ by the binary search
compute $\tau_0, \dots, \tau_{q+1}$, $q_\eta$, and $\xi_0, \dots, \xi_{r+1}$;
$y \leftarrow$ output of the continuous greedy with stopping time $1/4$ applied to (2);
$I' \leftarrow \emptyset$;
for $i \in I$ do
    choose a number $k$ from $[q]$ with probability $y_{ki}$ or $I' \leftarrow I' \cup \{i\}$ with probability
    $1 - \sum_{k \in [q]} y_{ki}$;
    if $i \notin I'$ then choose an integer $t_i$ from $[\tau_k, \tau_{k+1})$ uniformly at random;
$\Pi' \leftarrow$ sequence obtained by sorting the items $i \in I \setminus I'$ in a non-decreasing order of $t_i$;
$\Pi \leftarrow (\Pi'_1)$, $l \leftarrow 2$;
for $i = 2, \dots, |\Pi'|$ do
    if $\sum_{j \in [i-1]} s(\Pi'_j) < t_{\Pi'_i}$ then $\Pi_l \leftarrow \Pi'_i$, $l \leftarrow l+1$ ;
    else $I' \leftarrow I' \cup \{\Pi'_i\}$ ;
append the items in $I'$ to the suffix of $\Pi$ arbitrarily, and return $\Pi$;
```

the $|I|$-dimensional vector the component of which corresponding to $i \in I$ is z_{ji}. Then, the compact relaxation is described as follows.

$$\begin{array}{lll}
\text{maximize} & \sum_{j=0}^{q} \bar{p}(\tau_j)(F(z_{j+1}) - F(z_j)) & \\
\text{subject to} & \sum_{k \in [r]} y_{ki} \le 1, & \forall i \in I, \\
& \sum_{i \in I} \sum_{\xi_k < \tau_j} y_{ki} \min\{s(i), \tau_j\} \le 2\tau_j, & \forall j \in \{1, \dots, q\}, \\
& z_{ji} = \sum_{\xi_{k+1}-1 \le \tau_j - s(i)} y_{ki} & \forall j \in [q], \\
& y_{ki} \ge 0, & \forall i \in I, \forall k \in [r].
\end{array} \tag{2}$$

Lemma 5. *The optimal objective value of* (2) *is not smaller than that of* (1).

Lemma 6. *From a feasible solution to* (2) *achieving the objective value θ, we can construct a feasible solution to* (1) *achieving the objective value of at least $(1-\epsilon)(\theta - \epsilon w)/2$.*

We now wrap up our algorithm. Our algorithm first applies the continuous greedy algorithm with stopping time $1/4$ to compute a solution y such that $4y$ is feasible for (2) and the objective value of y in (2) is $1 - 1/\sqrt[4]{e}$ times that of any feasible solution, particularly the optimal value OPT of (2). From y, we compute a solution x for (1) by Lemma 6. $4x$ is feasible for (1), and the objective value of x in (1) is at least $(1-\epsilon)((1-1/\sqrt[4]{e})OPT - \epsilon w)/2$. Since we are assuming $\bar{p}(\min_{i \in I} s(i)) \ge 1$, picking the item of the smallest size at time 0 achieves objective value w. This means $OPT \ge w$, and hence the objective value of x is at least $(1-\epsilon)(1-\epsilon-1/\sqrt[4]{e})/2 \cdot OPT$. Then, applying the rounding algorithm to $4x$, we obtain a sequence of objective value $(1-\epsilon)(1-1/\sqrt[4]{e}-\epsilon)/4 \cdot OPT$.

To make this algorithm run in polynomial-time, we do not explicitly write down x. In the rounding algorithm, values of x are used for deciding t_i for each $i \in I$ in the first round of the rounding algorithm. This is possible without writing down x as follows. Notice that x_{ti} takes the same value for any $t \in [\tau_j, \tau_{j+1})$ by the construction of x. Hence each i chooses t_i as follows. First, it chooses $k \in [q]$ with probability y_{ki}, and is discarded with probability $1 - \sum_{k \in [r]} y_{ki}$. Then, it chooses t_i from $[\tau_k, \tau_{k+1})$ uniformly at random. This algorithm runs in polynomial time with respect to $1/\epsilon$ and the input size of the instance. We give a pseudo-code of the algorithm in Algorithm 5.

With the conversion given above, we obtain the following theorem.

Theorem 5. *For any constant $\epsilon \in (0, 1)$, there exists a randomized approximation algorithm of approximation ratio $(1-\epsilon)(1-\epsilon-1/\sqrt[4]{e})/4 \approx 0.055 - \epsilon$ for the stochastic capacity and no cancellation setting, which runs in polynomial time with respect to $1/\epsilon$ and the input size of the instance.*

Acknowledgement. The first author is supported by JSPS KAKENHI Grant Number JP16K16005. The second author is supported by JSPS KAKENHI Grant Number JP17K12646 and JST ERATO Grant Number JPMJER1201, Japan. The third author is supported by JSPS KAKENHI Grant Number JP17K00040 and JST ERATO Grant Number JPMJER1201, Japan.

References

1. Adamczyk, M., Sviridenko, M., Ward, J.: Submodular stochastic probing on matroids. Math. Oper. Res. **41**(3), 1022–1038 (2016)
2. Asadpour, A., Nazerzadeh, H., Saberi, A.: Stochastic submodular maximization. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 477–489. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92185-1_53
3. Călinescu, G., Chekuri, C., Pál, M., Vondrák, J.: Maximizing a monotone submodular function subject to a matroid constraint. SIAM J. Comput. **40**(6), 1740–1766 (2011)
4. Dabney, M.: A PTAS for the uncertain capacity knapsack problem. Master's thesis, Clemson University (2010)
5. Dean, B.C., Goemans, M.X., Vondrák, J.: Approximating the stochastic knapsack problem: the benefit of adaptivity. Math. Oper. Res. **33**(4), 945–964 (2008)
6. Disser, Y., Klimm, M., Megow, N., Stiller, S.: Packing a knapsack of unknown capacity. SIAM J. Discrete Math. **31**(3), 1477–1497 (2017)
7. Feldman, M.: Maximization problems with submodular objective functions. Ph.D. thesis, Technion - Israel Institute of Technology, July 2013
8. Feldman, M., Naor, J., Schwartz, R.: A unified continuous greedy algorithm for submodular maximization. In: FOCS, pp. 570–579 (2011)
9. Feldman, M., Svensson, O., Zenklusen, R.L.: Online contention resolution schemes. In: SODA, pp. 1014–1033 (2016)
10. Golovin, D., Krause, A.: Adaptive submodularity: theory and applications in active learning and stochastic optimization. J. Artif. Intell. Res. **42**, 427–486 (2011)
11. Gupta, A., Krishnaswamy, R., Molinaro, M., Ravi, R.: Approximation algorithms for correlated knapsacks and non-martingale bandits. In: FOCS, 827–836 (2011)
12. Gupta, A., Nagarajan, V., Singla, S.: Adaptivity gaps for stochastic probing: submodular and XOS functions. In: SODA, pp. 1688–1702 (2017)
13. Höhn, W., Jacobs, T.: On the performance of Smith's rule in single-machine scheduling with nonlinear cost. ACM Trans. Algorithms **11**(4), 25:1–25:30 (2015)

14. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective outbreak detection in networks. In: KDD, pp. 420–429 (2007)
15. Ma, W.: Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms: extended abstract. In: SODA, pp. 1154–1163 (2014)
16. Megow, N., Mestre, J.: Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In: ITCS, pp. 495–504 (2013)
17. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 745–756. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39206-1_63
18. Sviridenko, M.: A note on maximizing a submodular set function subject to a knapsack constraint. Oper. Res. Lett. **32**(1), 41–43 (2004)

Select and Permute: An Improved Online Framework for Scheduling to Minimize Weighted Completion Time

Samir Khuller[1], Jingling Li[1], Pascal Sturmfels[2], Kevin Sun[3], and Prayaag Venkat[1](✉)

[1] University of Maryland, College Park, MD 20742, USA
samir@cs.umd.edu, jinglingli1024@gmail.com, pkvasv@gmail.com
[2] University of Michigan, Ann Arbor, MI 48109, USA
psturm@umich.edu
[3] Duke University, Durham, NC 27708, USA
ksun@cs.duke.edu

Abstract. In this paper, we introduce a new online scheduling framework for minimizing total weighted completion time in a general setting. The framework is inspired by the work of Hall et al. [10] and Garg et al. [8], who show how to convert an offline approximation to an online scheme. Our framework uses two offline approximation algorithms—one for the simpler problem of scheduling without release times, and another for the *minimum unscheduled weight problem*—to create an online algorithm with provably good competitive ratios.

We illustrate multiple applications of this method that yield improved competitive ratios. Our framework gives algorithms with the best or only-known competitive ratios for the concurrent open shop, coflow, and concurrent cluster models. We also introduce a randomized variant of our framework based on the ideas of Chakrabarti et al. [3] and use it to achieve improved competitive ratios for these same problems.

Keywords: Coflow scheduling · Concurrent clusters
Concurrent open shop · Online algorithms

1 Introduction

Modern computing frameworks such as MapReduce, Spark, and Dataflow have emerged as essential tools for big data processing and cloud computing. In order to exploit large-scale parallelism, these frameworks act in several computation stages, which are interspersed with intermediate data transfer stages. During data transfer, results from computations must be efficiently scheduled for transfer across clusters so that the next computation stage can begin.

All authors performed this work at the University of Maryland, College Park, under the support of NSF REU Grant CNS 156019. We would also like to thank An Zhu and Google for their support, and the LILAC program at Bryn Mawr College.

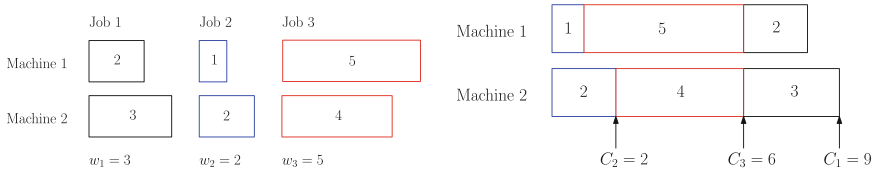
The coflow model [5,6] and the concurrent cluster model [11,19] were introduced to capture the distributed processing requirements of jobs across many machines. In these models, the objective of primary theoretical and practical interest is to minimize average job completion time [1,5,6,14,19,20]. The *concurrent open shop problem*, a special case of the above models, has emerged as a key subroutine for designing better approximation algorithms [1,14,19].

There has been a lot of work studying offline algorithms for these problems (see [1,14,20] for the coflow model, [19] for the concurrent cluster model, and [4,8,18,23] for the concurrent open shop model), but in real-world applications, jobs often arrive in an online fashion, so studying online algorithms is critical for accurate modeling of data centers.

Hall et al. [10] proposed a general framework which converts offline scheduling algorithms to online ones. Inspired by this result, we introduce a new online framework that improves upon the online algorithms of Garg et al. [8] for concurrent open shop and also gives the first algorithms with constant competitive ratios for other multiple-machine scheduling settings.

1.1 Formal Problem Statement

In the concurrent open shop setting, the problem is to schedule a set of jobs with machine-dependent components on a set of machines. Let $J = \{1, \ldots, n\}$ denote the set of jobs and $M = \{1, \ldots, m\}$ denote the set of machines. Each job j has one component for each of the m machines. For each job j, we denote the processing time of the component on machine i as p_{ij}, its release time as r_j, and its weight as w_j. The different components of each job can be processed concurrently and in any order, as long as no component of job j is processed before r_j. Job j is complete when all of its components have been processed; we denote its completion time by C_j. Our goal is to specify a schedule of the jobs on the machines that minimizes $\sum_{j \in J} w_j C_j$; see Fig. 1 for an example.



(a) An instance of concurrent open shop. (b) The optimal schedule

Fig. 1. All jobs are released at the same time, and the processing requirement for each job-machine combination is specified inside the blocks.

We follow the 3-field $\alpha\,|\beta|\,\gamma$ notation (see [9]) for scheduling problems, where α denotes the scheduling environment, β denotes the job characteristics, and γ denotes the objective function. As stated above, we focus on the case where $\gamma = \sum_j w_j C_j$. In accordance with the notation of [10,18,19], we let $\alpha = PD$

denote the concurrent open shop setting and $\alpha = CC$ denote the concurrent cluster setting, see below for definitions.

1.2 Related Work

The concurrent open shop model is a relaxation of the well-known open shop model that allows components of the same job to be processed in parallel on different machines. Roemer [21] showed that $PD \,||\, \sum_j w_j C_j$ is NP-hard and after several successive approximation hardness results [2,18], Sachdeva and Saket [22] showed that it is not approximable within a factor less than 2 unless P = NP, even when job release times are identical. For this model, Wang and Cheng [23] gave a $\frac{16}{3}$-approximation algorithm. This was later improved to a 2-approximation for identical job release times [4,8,15,18], matching the above lower bound, and a 3-approximation for arbitrary job release times [1,8,15]. In the preemptive setting, Im and Purohit [12] gave a $(2+\epsilon)$-approximation for arbitrary job release times.

In the online setting, Hall et al. [10] introduced a general framework that improved the best-known approximation guarantees for several well-studied scheduling environments. They showed that the existence of an offline *dual* ρ-approximation yields an online 4ρ-approximation, where a dual ρ-approximation is an algorithm that packs as much weight of jobs into a time interval of length ρD as the optimal algorithm does into an interval of length D. Furthermore, they showed that when $m = 1$, a local greedy ordering of jobs yields further improvements.

While the framework of Hall et al. [10] is entirely deterministic, Chakrabarti et al. [3] gave a randomized variant with an improved competitive ratio guarantee. Specifically, they showed a dual-ρ approximation algorithm can be converted to an expected 2.89ρ-competitive online scheduling algorithm in the same setting, improving upon the 4ρ competitive ratio of Hall et al. [10].

The online version of $PD \,||\, \sum_j w_j C_j$ was first studied by Garg et al. [8]. They noted that applying the framework of [10] was not straightforward, so they focused on minimizing the weight of unscheduled jobs rather than maximizing the weight of scheduled jobs. Using a similar approach to that of Hall et al. [10], they gave an exponential-time 4-competitive algorithm and a polynomial-time 16-competitive algorithm for the online version of $PD \,||\, \sum_j w_j C_j$.

The coflow scheduling model was first introduced as a networking abstraction to model communications in datacenters [5,6]. In the coflow scheduling problem, the goal is to schedule a set of *coflows* on a non-blocking switch with m input ports and m output ports, where any unused input-output ports can be connected via a path through unused nodes regardless of other existing paths. Each coflow is a collection of parallel flow demands that specify the amount of data that needs to be transferred from an input port to an output port.

For Coflow$\,|r_j = 0|\sum_j w_j C_j$, Qiu et al. [20] gave deterministic $\frac{64}{3}$ and randomized $(8 + \frac{16\sqrt{2}}{3})$ approximation algorithms. For arbitrary release times, they gave deterministic $\frac{67}{3}$ and randomized $9 + \frac{16\sqrt{2}}{3}$ approximation algorithms. Khuller and Purohit [14] later improved these deterministic approximations to 8 and 12 for identical and arbitrary release times respectively, and also gave a randomized $(3+2\sqrt{2})$-approximation algorithm for identical release times. Ahmadi et al. [1] gave a deterministic 4-approximation and 5-approximation for identical and arbitrary release times, respectively. Recently, Im and Purohit [12] achieved a tight approximation ratio of $2+\epsilon$ for arbitrary release times. To the best of our knowledge, there are no known constant-factor competitive algorithms for online coflow scheduling, although Li et al. [16] give a $O(m \ln n)$-competitive algorithm when all coflow weights are equal to 1, where m is the number of coflows and n is the number of nodes in the network.

Finally, we mention the *concurrent cluster* model recently introduced by Murray et al. [19]. The concurrent cluster model generalizes the concurrent open shop model by replacing each machine by a cluster of machines, where different machines in the same cluster may have different processing speeds. Each job still has m processing requirements, but this requirement can be fulfilled by any machine in the corresponding cluster. Murray et al. [19] give the first constant-factor approximations for minimizing total weighted completion time via a reduction to concurrent open shop and a list-scheduling subroutine.

1.3 Paper Outline and Results

In Sect. 2, we introduce a general framework for designing online scheduling algorithms for minimizing total weighted completion time. The framework divides time into intervals of geometrically-increasing size, and greedily "packs" jobs into each interval, and then imposes a locally-determined ordering of the jobs within each interval. It is inspired by the framework of Hall et al. [10] and an adaptation by Garg et al. [8].

In Sect. 3, we apply our framework to $PD \,||\, \sum_j w_j C_j$. We show that an offline exponential-time algorithm that optimally solves $PD \,|r_j = 0|\, \sum_j w_j C_j$ yields an exponential-time 3-competitive algorithm for $PD \,||\, \sum_j w_j C_j$. We also combine the algorithms given by Garg et al. [8] and Mastrolilli et al. [18] to create a polynomial-time 10-competitive algorithm for $PD \,||\, \sum_j w_j C_j$. We conclude Sect. 3 by giving a polynomial-time $(3+\epsilon)$-competitive algorithm when the number of machines m is fixed. Details on the subroutines used in this section are provided in the full version of this paper [13].

In Sect. 4, extending the ideas of Sect. 3, we apply our framework to online coflow scheduling to design an exponential-time $(4+\epsilon)$-competitive algorithm, and a polynomial-time $(10+\epsilon)$-competitive algorithm.

Section 5 describes an extension of the techniques of Chakrabarti et al. [3] that produces a randomized variant of our framework that yields better competitive ratio guarantees than the deterministic version. The full version of this paper [13] describes the concurrent cluster model of Murray et al. [19]; we show that extending subroutines used for the concurrent open shop setting yields an online 19-competitive algorithm via our framework.

Table 1. A summary of online approximation guarantees and the best-known previous results, where m denotes the number of machines, ϵ is arbitrarily small, and "-" indicates the absence of a relevant result. The two numbers in each entry of the "Our ratios" column denote the competitive and expected ratio of our deterministic and randomized algorithms, respectively.

Problem	Running time	Our ratios	Previous ratio
$PD \,\|\| \sum_j w_j C_j$	Polynomial	10, 7.78	16 [8]
$PD \,\|\| \sum_j w_j C_j$	Exponential	3, 2.45	4 [8]
$PD \,\|\| \sum_j w_j C_j$	Polynomial, fixed m	$3+\epsilon$, $2.45+\epsilon$	-
Coflow $\|\| \sum_j w_j C_j$	Polynomial	$10+\epsilon$, $7.78+\epsilon$	-
Coflow $\|\| \sum_j w_j C_j$	Exponential	$4+\epsilon$, $3.45+\epsilon$	-
$CC \,\|\| \sum_j w_j C_j$	Polynomial	19, 14.55	-

2 A Minimization Framework for Online Scheduling

In this section, we introduce our framework for online scheduling problems. To motivate the key ideas of this section, we begin by briefly reviewing the work of Hall et al. [10] and Garg et al. [8].

2.1 The Maximization Framework of Hall et al. [10]

The framework of Hall et al. [10] divides the online problem into a sequence of offline *maximum scheduled weight* problems, each of which is solved using an offline *dual* approximation algorithm.

Definition 1 (Maximum scheduled weight problem (MSWP) [10]). *Given a set of jobs, a non-negative weight for each job, and a deadline D, construct a schedule that maximizes the total weight of jobs completed by time D.*

Definition 2 (Dual ρ-approximation algorithm [10]). *An algorithm for the MSWP is a* dual *ρ-approximation algorithm if it constructs a schedule of length at most ρD and has total weight at least that of the schedule which maximizes the weight of jobs completed by D.*

Fix a scheduling environment and suppose we have a dual ρ-approximation for the MSWP. We divide time into intervals of geometrically-increasing size by letting $t_0 = 0$ and $t_k = 2^{k-1}$ for $k = 1, \ldots, L$ where L is large enough to cover the entire time horizon. At each time t_k, let $R(t_k)$ denote the set of jobs that have arrived by t_k but have not yet been scheduled. We run the dual ρ-approximation algorithm on $R(t_k)$ with deadline $D = t_{k+1} - t_k = t_k$. In the output schedule, we take only jobs which complete by ρD and schedule them in the interval starting at time ρt_k. Hall et al. [10] show that this framework produces an online 4ρ-competitive algorithm.

2.2 The Minimum Unscheduled Weight Problem of Garg et al. [8]

Garg et al. [8] sought to apply the framework of Hall et al. [10] to the concurrent open shop setting. They noted that devising a dual-ρ approximation algorithm for concurrent open shop was difficult, so they instead proposed a variant of the MSWP. The definitions below generalize those used by Garg et al. [8] to arbitrary scheduling problems.

Definition 3 (Minimum unscheduled weight problem (MUWP)). *Given a set of jobs, a non-negative weight for each job, and a deadline D, find a subset of jobs S which can be completed by time D and minimizes the total weight of jobs not in S. We call this quantity the* unscheduled weight*.*

Definition 4 ((α, β)-approximation algorithm). *An algorithm for the MUWP is an* (α, β)-approximation *if it finds a subset of jobs which can be completed by αD and has unscheduled weight at most β times that of the subset of jobs with minimum unscheduled weight that completes by D.*

Note that a dual ρ-approximation for the MSWP is a $(\rho, 1)$-approximation for the MUWP. With these definitions, Garg et al. [8] established constant-factor approximations for $PD \,||\, \sum_j w_j C_j$.

2.3 A Minimization Framework

We now describe a new framework inspired by the ideas of Hall et al. [10] and Garg et al. [8]. For the settings we consider, previous online algorithms do not impose any particular ordering of jobs within each interval, which can lead to schedules with poor local performance. In our framework, we make use of a γ-approximation to the offline version of the scheduling problem with identical release times to address this issue.

As in the works of Hall et al. [10] and Garg et al. [8], we assume that all processing times are at least 1. This is to avoid the extreme scenario that a single job of size $\epsilon \ll 1$ arrives just after time 0, and our framework waits until time 1 to schedule, thus leading to arbitrarily large competitive ratio.

Let W denote the total weight of all the jobs in J, and let $W_\tau^{\mathcal{A}}$ ($W_\tau^{\mathcal{OPT}}$) denote the total weight of jobs that complete after time τ by our algorithm $\mathcal{A}$ (by the optimal algorithm $\mathcal{OPT}$). Note that $W_\tau^{\mathcal{A}}, W_\tau^{\mathcal{OPT}}$ include the weight of jobs not yet released at time τ. Let $\tau_0 = 0$, and for $k \geq 1$, let $\tau_k = 2^{k-1}$, I_k denote the k^{th} interval $[\tau_k, \tau_{k+1})$, αI_k denote $[\alpha\tau_k, \alpha\tau_{k+1})$, and $R(\tau_k)$ denote the set of jobs released but not yet scheduled before τ_k by $\mathcal{A}$.

Our online algorithm $\mathcal{A}$ works as follows. At each τ_k, run an (α, β)-approximation algorithm on $R(\tau_k)$ with deadline $D = \tau_{k+1} - \tau_k$. Schedule the output set of jobs in αI_k using the offline γ-approximation algorithm.[1]

Theorem 1. *Algorithm $\mathcal{A}$ is $(2\alpha\beta + \gamma)$-competitive, with an additive αW term.*

To prove Theorem 1, we first show that at each time step, $\mathcal{A}$ remains competitive with the optimal schedule by incurring a time delay.

Lemma 1. *For any $k \geq 0$, we have $W^{\mathcal{A}}_{\alpha\tau_{k+1}} \leq \beta W^{\mathcal{OPT}}_{\tau_k}$.*

Proof. Every job completed by $\mathcal{OPT}$ by τ_k must have been released before τ_k. For each such job j, either our algorithm completed it before time τ_k or $j \in R(\tau_k)$. The set of jobs completed by $\mathcal{OPT}$ by time τ_k gives a feasible solution to the MUWP with deadline $D = \tau_{k+1} - \tau_k = \tau_k$ and its total unscheduled weight is $W^{\mathcal{OPT}}_{\tau_k}$. Therefore, the optimal total unscheduled weight value for the MUWP when considering all $j \in R(\tau_k)$ with deadline D is at most $W^{\mathcal{OPT}}_{\tau_k}$. By the definition of (α, β)-approximation, the claim follows. $\square$

The next lemma states that ordering jobs within each interval further approximates the optimal schedule closely. For a fixed subset S of jobs, let $\mathcal{OPT}(S)$ denote the optimal schedule for S and $C_j^{\mathcal{OPT}(S)}$ denote the completion time of job j in $\mathcal{OPT}(S)$. Also, let $\mathcal{OPT}_0(S)$ denote an optimal schedule that starts at time 0 and ignores all job release times, and let $C_j^{\mathcal{OPT}_0(S)}$ denote the completion time of job j in $\mathcal{OPT}_0(S)$.

Lemma 2. *The weighted completion time for schedule $\mathcal{OPT}_0(S)$ is at most that of schedule $\mathcal{OPT}(S)$; i.e.,*

$$\sum_{j \in S} w_j C_j^{\mathcal{OPT}_0(S)} \leq \sum_{j \in S} w_j C_j^{\mathcal{OPT}(S)}.$$

Proof. The optimal schedule of S with release times defines a valid schedule for S without release times, so the claim follows. $\square$

Recall that at each τ_k, $\mathcal{A}$ uses an (α, β)-approximation on the MUWP to select a subset S_k of $R(\tau_k)$ to schedule within αI_k using a γ-approximation that ignores release times. Let $C_j^{\mathcal{A}}$ denote the completion time of job j in the schedule produced by $\mathcal{A}$, $t(j)$ denote the largest index such that job j begins processing

[1] We make the critical assumption that the offline γ-approximation algorithm does not increase the makespan of the given subset of jobs, so as to ensure that the schedule fits inside of αI_k. For the scheduling models studied in this paper, this assumption will indeed hold. In fact, if it can be shown that the γ-approximation algorithm also approximates the makespan criteria within some factor μ, then it is straightforward to incorporate this into the model, at the expense of an additional μ factor in the approximation guarantee. For example, Chakrabarti et al. [3] provide bicriteria approximation algorithms for the total weighted completion time and makespan objective functions.

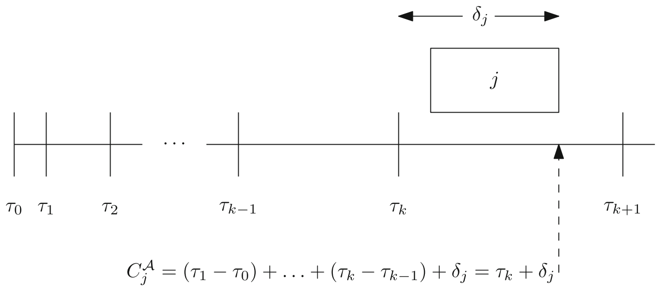

δ_j
j
τ_0
τ_1
τ_2
τ_{k-1}
τ_k
τ_{k+1}
$C_j^{\mathcal{A}} = (\tau_1 - \tau_0) + \ldots + (\tau_k - \tau_{k-1}) + \delta_j = \tau_k + \delta_j$

Fig. 2. We let δ_j denote the distance between $C_j^{\mathcal{A}}$ and the beginning of the interval in which job j completes.

after time $\tau_{t(j)}$, and $\delta_j = C_j^{\mathcal{A}} - \alpha\tau_{t(j)}$ for each job $j \in J$ (see Fig. 2). Let L be the smallest time index such that the optimal schedule finishes by time τ_L, and let S_k denote the set of jobs scheduled independently by $\mathcal{A}$ in the interval αI_k. Then Lemma 2 implies

$$\sum_{j\in S_k} w_j\delta_j \le \gamma \sum_{j\in S_k} w_j C_j^{\mathcal{OPT}_0(S_k)} \le \gamma \sum_{j\in S_k} w_j C_j^{\mathcal{OPT}(S_k)}. \tag{1}$$

Lemma 3. *The weighted sum of the δ_j is at most γ times the optimal weighted completion time; i.e.,*

$$\sum_{j\in J} w_j\delta_j \le \gamma \sum_{j\in J} w_j C_j^{\mathcal{OPT}}.$$

Proof. Recall that $S_1, \ldots, S_L$ partition J, and notice that due to (1), we have

$$\sum_{k=1}^{L}\sum_{j\in S_k} w_j\delta_j \le \gamma \sum_{k=1}^{L}\sum_{j\in S_k} w_j C_j^{\mathcal{OPT}(S_k)} \le \gamma \sum_{j\in J} w_j C_j^{\mathcal{OPT}}, \tag{2}$$

thus proving the lemma. □

Proof (of Theorem 1). We rewrite the total weighted completion time of the schedule produced by $\mathcal{A}$ to obtain the following.

$$\begin{aligned}\sum_{j\in J} w_j C_j^{\mathcal{A}} &= \alpha \sum_{k=1}^{L} (\tau_k - \tau_{k-1}) W_{\alpha\tau_k}^{\mathcal{A}} + \sum_{j\in J} w_j\delta_j \\ &= \alpha \sum_{k=2}^{L} (\tau_k - \tau_{k-1}) W_{\alpha\tau_k}^{\mathcal{A}} + \alpha W_{\alpha\tau_1}^{\mathcal{A}} + \sum_{j\in J} w_j\delta_j \\ &\le 2\alpha \sum_{k=1}^{L} (\tau_k - \tau_{k-1}) W_{\alpha\tau_{k+1}}^{\mathcal{A}} + \alpha W + \sum_{j\in J} w_j\delta_j\end{aligned}$$

$$\leq 2\alpha\beta \sum_{k=1}^{L} (\tau_k - \tau_{k-1}) W_{\tau_k}^{\mathcal{OPT}} + \alpha W + \sum_{j \in J} w_j \delta_j$$
$$\leq (2\alpha\beta + \gamma) \sum_{j \in J} w_j C_j^{\mathcal{OPT}} + \alpha W,$$

where the last two inequalities follow from Lemmas 1 and 3, respectively. □

3 Applications to Concurrent Open Shop

Now we apply our minimization framework to $PD \,||\, \sum_j w_j C_j$. In the full version of this paper [13], we give an offline dynamic program that optimally solves $PD \,|r_j = 0|\, \sum_j w_j C_j$ in exponential time, giving $\gamma = 1$ in our framework.

For the MUWP, in exponential time, we can iterate over every subset of jobs to find a feasible schedule that minimizes the total weight of unscheduled jobs, so this is a $(1, 1)$-approximation, giving $\alpha = \beta = 1$. Thus, Theorem 1 yields the following, which improves upon the competitive ratio of 4 from Garg et al. [8].

Corollary 1. *There exists an exponential time 3-competitive algorithm for the concurrent open shop setting.*

In polynomial time, Garg et al. [8] provide a $(2, 2)$-approximation for the MUWP, and Mastrolilli et al. [18] provide a 2-approximation for offline version of $PD \,|r_j = 0|\, \sum_j w_j C_j$. These results with Theorem 1 improve the ratio of 16 by Garg et al. [8]. We note that the additional additive term of αW in Theorem 1 is smaller than the additive $3W$ term in the guarantees of Garg et al. [8], for both the exponential-time and polynomial-time cases.

Corollary 2. *There exists a polynomial-time 10-competitive algorithm for the concurrent open shop setting.*

When the number of machines m is constant, there exists a polynomial-time $(1 + \epsilon, 1)$-approximation algorithm for the MUWP (see full version of this paper [13]) by a reduction to the multidimesional knapsack problem. Furthermore, when m is fixed, Edwin Cheng et al. [7] gave a PTAS for the offline $PD \,|r_j = 0|\, \sum_j w_j C_j$. Combining these results with Theorem 1 yields the following.

Corollary 3. *There exists a polynomial time, $(3 + \epsilon)$-competitive algorithm for $PD \,||\, \sum w_j C_j$ when the number of machines is fixed.*

4 Applications to Coflow Scheduling

We now apply our framework to coflow scheduling, introduced by Chowdhury and Stoica [5]. We are given a non-blocking network with m input ports and m output ports. A *coflow* is a collection of parallel flows processed by the network.

We represent a coflow j by an $m \times m$ matrix $D^j = (d^j_{io})_{i,o \in [m]}$, where d^j_{io} denotes the integer amount of data to be transferred from input port i to output port o for coflow j. Each port can process at most one unit of data per time unit, and we assume that the transfer of data within the network is instantaneous.

The problem is to schedule a set of n coflows, each with a non-negative weight w_j and release time r_j, that minimizes the sum of weighted completion times, where the completion time of a coflow is the earliest time at which all of its flows have been processed. We denote this problem by Coflow $|| \sum_j w_j C_j$.

As in Sect. 3, in exponential time, we can iterate over all subsets of coflows to optimally solve the MUWP, giving a $(1, 1)$-approximation. Moreover, Im and Purohit [12] proposed a $(2 + \epsilon)$-approximation for offline coflow scheduling[2].

Corollary 4. *There exists an exponential-time* $(4+\epsilon)$*-competitive algorithm for online coflow scheduling.*

Furthermore, we can show that the polynomial-time $(2, 2)$-approximation for the MUWP for $PD || \sum_j w_j C_j$ of Garg et al. [8] can be applied to coflow scheduling with the same approximation guarantees. Combined with the 4-approximation of Ahmadi et al. [1], our framework yields the following.

Corollary 5. *There exists a polynomial-time* $(10+\epsilon)$*-competitive algorithm for online coflow scheduling.*

To show that the $(2, 2)$-approximation for the MUWP for $PD || \sum_j w_j C_j$ of Garg et al. [8] can be applied to coflow scheduling with the same approximation guarantees, we recall the reduction from Coflow $|| \sum_j w_j C_j$ to $PD || \sum_j w_j C_j$ given by Khuller and Purohit [14]. Given an instance of coflow scheduling $\mathcal{I}$, let $L^j_i = \sum_{o=1}^m d^j_{io}$ denote the total amount of data that co-flow j needs to transmit through input port i, and similarly, we let $L^j_o = \sum_{i=1}^m d^j_{io}$ or output port o. From this, create a concurrent open shop instance $\mathcal{I}'$ with a set M of $2m$ machines (one for each port) and a set J of n jobs (one for each coflow), with processing times p_{sj} set equal to L^j_s for job j on machine s.

Now, the MUWP on $\mathcal{I}'$ can be formulated by the following integer program of Garg et al. [8].

$$\begin{aligned} \text{minimize} \; & \sum_{j \in J} w_j (1 - x_j) \\ \text{subject to} \; & \sum_{j \in J} p_{ij} x_j \leq D \quad \forall i \in M \\ & x_j \in \{0, 1\} \quad \forall j \in J. \end{aligned}$$

Let W' denote the optimal unscheduled weight for the MUWP on $\mathcal{I}'$, and define W similarly. The algorithm of Garg et al. [8] solves the linear relaxation

[2] Since permutation schedules are not necessarily optimal for coflow scheduling [6], even finding a *factorial*-time optimal algorithm is nontrivial. For simplicity, we have chosen to use a polynomial-time algorithm to achieve Corollary 4.

of this integer program to obtain an optimal fractional solution $\bar{x}$, and outputs the set of jobs $S' = \{j \in J \mid \bar{x}_j \geq \frac{1}{2}\}$. Letting W^* denote the objective function value of an optimal solution of the LP relaxation, it is straightforward to check that the total processing time of S' on any machine is at most $2D$, the total unscheduled weight is at most $2W^*$, and $W^* \leq W'$. Hence, the algorithm of Garg et al. [8] is indeed a $(2, 2)$-approximation for the MUWP in the concurrent open shop environment.

Lemma 4. *The optimal unscheduled weight for the MUWP on $\mathcal{I}'$ is at most that for the MUWP on $\mathcal{I}$; i.e., $W' \leq W$.*

Proof. The proof is essentially identical to that of Lemma 1 in [14]. Let S be the optimal solution to the MUWP for $\mathcal{I}$. Then there exists a schedule of the coflows in S such that every coflow completes by the deadline D. Now consider the set S' of corresponding jobs in $\mathcal{I}'$. Processing job $j \in S'$ on machine s whenever data is being processed for coflow $j \in S$ on port s in the schedule for S gives a schedule for S' in which every job also completes by deadline D. Thus S' is a feasible solution to the MUWP for $\mathcal{I}'$ with objective function value equal to that of the optimal solution S to the MUWP for $\mathcal{I}$, and the claim follows. □

Let S be the set of coflows in $\mathcal{I}$ corresponding to the jobs S' defined above.

Lemma 5. *In polynomial time, we can find a schedule for S that completes by time $2D$, and whose total unscheduled weight is at most $2W$.*

Proof. We know that for any machine s in $\mathcal{I}'$, $\sum_{j \in S'} p_{sj} = \sum_{j \in S} L_s^j \leq 2D$. Thus, if we take any schedule for the coflows S without idle time, every port s finishes processing data by time $\sum_{j \in S} L_s^j \leq 2D$. Since all coflows complete at the same time when all ports have finished processing the data, we get a schedule in which all coflows in S will complete without idle time by time $2D$.

The total unscheduled weight in $\mathcal{I}$ is the same as the total unscheduled weight in $\mathcal{I}'$. By Lemma 4, the total unscheduled weight in $\mathcal{I}$ is at most $2W' \leq 2W$. □

Hence, the $(2, 2)$-approximation for the MUWP for $PD \,||\, \sum_j w_j C_j$ of Garg et al. [8] can be applied to Coflow $||\, \sum_j w_j C_j$ with the same guarantees.

5 A Randomized Online Scheduling Framework

In this section, we show how our ideas can be combined with the randomized framework of Chakrabarti et al. [3] to develop an analogue of the deterministic framework of Sect. 2.

The framework of Chakrabarti et al. [3] modify that of Hall et al. [10] (see Sect. 2.1) by setting $\tau_k = \eta 2^k$, where $\eta \in [\frac{1}{2}, 1)$ is a randomly chosen parameter. After making this choice, the online algorithm proceeds exactly as before, by applying the dual ρ-approximation to the MSWP at each interval.

Let $C_j^{\mathcal{OPT}}$ denote the completion time of job j in an optimal schedule, and let B_j denote the start of the interval $(\tau_{k-1}, \tau_k]$ in which job j completes. Chakrabarti et al. [3] show that if one takes $\eta = 2^{-X}$, where X is chosen uniformly at random from $(0, 1]$, then the following holds.

Lemma 6 **([3]).** $E[B_j] = \frac{1}{2\ln 2} C_j^{\mathcal{OPT}}$.

Hall et al. [10] showed how to produce a schedule of total weighted completion time at most $4\rho \sum_j w_j B_j$. By linearity of expectation, one can apply Lemma 6, so that the schedule produced has total weighted completion at most $\frac{2}{\ln 2}\rho \sum_j w_j C_j^{\mathcal{OPT}}$, resulting in a randomized $\frac{2}{\ln 2}\rho \leq 2.89\rho$-competitive algorithm.

We can directly adapt this idea of randomly choosing the interval end points in our minimization framework to develop a randomized version of Theorem 1. Specifically, we take $\tau_k = \eta 2^k$, using the same η above, and run the framework described in Sect. 2.3 using this new choice of interval end points. Note that Lemmas 1, 2, and 3 still hold with our new choice of τ_k.

Let $\mathcal{A}'$ denote this randomized algorithm and using the same notation as in Sect. 2.3, we can achieve the following result.

Theorem 2. *Algorithm $\mathcal{A}'$ is $(\frac{1}{\ln 2}\alpha\beta + \gamma)$-competitive in expectation, with an additive αW term.*

Proof. The same steps as in the proof of Theorem 1 yield

$$\sum_{j \in J} w_j C_j^{\mathcal{A}'} \leq 2\alpha\beta \sum_{k=1}^{L} (\tau_k - \tau_{k-1}) W_{\tau_k}^{\mathcal{OPT}} + \alpha W + \sum_{j \in J} w_j \delta_j.$$

By definition of B_j, we notice that

$$\sum_{k=1}^{L} (\tau_k - \tau_{k-1}) W_{\tau_k}^{\mathcal{OPT}} = \sum_{j \in J} w_j B_j.$$

By linearity of expectation and Lemma 6, we conclude that

$$E[\sum_{j \in J} w_j C_j^{\mathcal{A}'}] \leq (\frac{1}{\ln 2}\alpha\beta + \gamma) \sum_{j \in J} w_j C_j^{\mathcal{OPT}} + \alpha W.$$

□

Using the guarantee of Theorem 2, we can instantiate this framework in various scheduling settings and find values for α, β, γ to achieve improved competitive ratios over our deterministic framework. The results obtained when applying the same subroutines as we did for the framework of Sect. 2.3 are in Table 1.

Acknowledgements. We would like to thank Sungjin Im and Clifford Stein for directing us to [3,17], and William Gasarch for organizing the REU program.

References

1. Ahmadi, S., Khuller, S., Purohit, M., Yang, S.: On scheduling coflows. In: Eisenbrand, F., Koenemann, J. (eds.) IPCO 2017. LNCS, vol. 10328, pp. 13–24. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59250-3_2
2. Bansal, N., Khot, S.: Inapproximability of hypergraph vertex cover and applications to scheduling problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 250–261. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14165-2_22
3. Chakrabarti, S., Phillips, C.A., Schulz, A.S., Shmoys, D.B., Stein, C., Wein, J.: Improved scheduling algorithms for minsum criteria. In: Meyer, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 646–657. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61440-0_166
4. Chen, Z.L., Hall, N.G.: Supply chain scheduling: assembly systems. Technical report. University of Pennsylvania (2000)
5. Chowdhury, M., Stoica, I.: Coflow: a networking abstraction for cluster applications. In: HotNets, pp. 31–36 (2012)
6. Chowdhury, M., Zhong, Y., Stoica, I.: Efficient coflow scheduling with Varys. In: ACM SIGCOMM CCR, vol. 44, pp. 443–454. ACM (2014)
7. Edwin Cheng, T., Nong, Q., Ng, C.T.: Polynomial-time approximation scheme for concurrent open shop scheduling with a fixed number of machines to minimize the total weighted completion time. Nav. Res. Logist. **58**(8), 763–770 (2011)
8. Garg, N., Kumar, A., Pandit, V.: Order scheduling models: hardness and algorithms. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 96–107. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77050-3_8
9. Graham, R., Lawler, E., Lenstra, J., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. **5**, 287–326 (1979)
10. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: off-line and on-line approximation algorithms. Math. Oper. Res. **22**(3), 513–544 (1997)
11. Hung, C.C., Golubchik, L., Yu, M.: Scheduling jobs across geo-distributed datacenters. In: SoCC, pp. 111–124. ACM (2015)
12. Im, S., Purohit, M.: A tight approximation for co-flow scheduling for minimizing total weighted completion time. arXiv preprint arXiv:1707.04331 (2017)
13. Khuller, S., Li, J., Sturmfels, P., Sun, K., Venkat, P.: Select and permute: an improved online framework for scheduling to minimize weighted completion time. arXiv preprint arXiv:1704.06677 (2017)
14. Khuller, S., Purohit, M.: Brief announcement: improved approximation algorithms for scheduling co-flows. In: SPAA, pp. 239–240. ACM (2016)
15. Leung, J.Y.T., Li, H., Pinedo, M.: Scheduling orders for multiple product types to minimize total weighted completion time. Discret. Appl. Math. **155**(8), 945–970 (2007)
16. Li, Y., Jiang, S.H.C., Tan, H., Zhang, C., Chen, G., Zhou, J., Lau, F.: Efficient online coflow routing and scheduling. In: Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 161–170. ACM (2016)
17. Lübbecke, E., Maurer, O., Megow, N., Wiese, A.: A new approach to online scheduling: approximating the optimal competitive ratio. ACM Trans. Algorithms (TALG) **13**(1), 15 (2016)

18. Mastrolilli, M., Queyranne, M., Schulz, A.S., Svensson, O., Uhan, N.A.: Minimizing the sum of weighted completion times in a concurrent open shop. Oper. Res. Lett. **38**(5), 390–395 (2010)
19. Murray, R., Chao, M., Khuller, S.: Scheduling distributed clusters of parallel machines: primal-dual and LP-based approximation algorithms. In: ESA (2016)
20. Qiu, Z., Stein, C., Zhong, Y.: Minimizing the total weighted completion time of coflows in datacenter networks. In: SPAA, pp. 294–303. ACM (2015)
21. Roemer, T.A.: A note on the complexity of the concurrent open shop problem. J. Sched. **9**(4), 389–396 (2006)
22. Sachdeva, S., Saket, R.: Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In: CCC, pp. 219–229. IEEE (2013)
23. Wang, G., Cheng, T.E.: Customer order scheduling to minimize total weighted completion time. Omega **35**(5), 623–626 (2007)

Recognizing Generalized Transmission Graphs of Line Segments and Circular Sectors

Katharina Klost(✉) and Wolfgang Mulzer

Institut für Informatik, Freie Universität Berlin, Berlin, Germany
{kathklost,mulzer}@inf.fu-berlin.de

Abstract. Suppose we have an arrangement $\mathcal{A}$ of n geometric objects $x_1, \dots, x_n \subseteq \mathbb{R}^2$ in the plane, with a distinguished point p_i in each object x_i. The *generalized transmission graph* of $\mathcal{A}$ has vertex set $\{x_1, \dots, x_n\}$ and a *directed* edge $x_i x_j$ if and only if $p_j \in x_i$. Generalized transmission graphs provide a generalized model of the connectivity in networks of directional antennas.

The complexity class $\exists\mathbb{R}$ contains all problems that can be reduced in polynomial time to an existential sentence of the form $\exists x_1, \dots, x_n : \phi(x_1, \dots, x_n)$, where $x_1, \dots, x_n$ range over $\mathbb{R}$ and ϕ is a propositional formula with signature $(+, -, \cdot, 0, 1)$. The class $\exists\mathbb{R}$ aims to capture the complexity of the *existential theory of the reals*. It lies between **NP** and **PSPACE**.

Many geometric decision problems, such as recognition of disk graphs and of intersection graphs of lines, are complete for $\exists\mathbb{R}$. Continuing this line of research, we show that the recognition problem of generalized transmission graphs of line segments and of circular sectors is hard for $\exists\mathbb{R}$. As far as we know, this constitutes the first such result for a class of *directed* graphs.

1 Introduction

Let $\mathcal{A}$ be an arrangement of n geometric objects $x_1, \dots, x_n$ in the plane. The *intersection graph* of $\mathcal{A}$ has one vertex for each object and an undirected edge between two objects x_i and x_j if and only if x_i and x_j intersect. In particular, if the objects are (unit) disks, we speak of *(unit) disk graphs*. These are often used as a symmetric model for antenna reachability. In some cases, however, this symmetry is not desired, since it does not accurately model the properties of the network. For omnidirectional antennas, there is an asymmetric model called *transmission graphs* [2]. Transmission graphs are also defined on disks: as in disk graphs, there is one vertex per disk, and the edges indicate directed reachability. There is a *directed* edge between two disks if the first disk contains the center of the second disk.

Here, we present a new class of *generalized transmission graphs*. Now, the objects may be arbitrary sets in $\mathbb{R}^2$, and the points that decide about the existence of an edge can be arbitrary points in the objects.

Supported in part by ERC StG 757609.

For a given graph class, the *recognition problem* is as follows: given a combinatorial graph $G = (V, E)$, decide whether G belongs to this class. For the recognition of geometrically defined graphs, it turned out that the complexity class $\exists\mathbb{R}$ plays a major role. The class $\exists\mathbb{R}$ was formally introduced by Schaefer [7]. It consists of all problems that are polynomial-time reducible to the set of all true sentences of the form $\exists x_1, \ldots, x_n : \Phi(x_1, \ldots, x_n)$. Here, Φ is a quantifier-free formula with signature $(+, -, \cdot, 0, 1)$ additional to the standard boolean signature. The variables range over the reals. Hardness for this class is defined via polynomial reduction.

There are multiple classes of intersection graphs for which the recognition problem is $\exists\mathbb{R}$-complete. Kang and Müller showed this for intersection graphs of k-spheres [1], and Schaefer proved a similar result for intersection graphs of line segments and convex sets [7].

One prototypical $\exists\mathbb{R}$-complete problem that serves as the starting point of many reductions is Stretchability, which was among the first known $\exists\mathbb{R}$-hard problems. The original hardness-proof is due to Mnëv [6], and it was restated in terms of $\exists\mathbb{R}$ by Matoušek [5].

Here, we show that the recognition of generalized transmission graphs of line segments and of a certain type of arrangements of circular sectors is hard for $\exists\mathbb{R}$. For this, we need to extend the known proofs significantly, and we need to develop new tools to reason about geometric realizations of directed graphs. With some further work the inclusion of these problems in $\exists\mathbb{R}$ could be shown. For details see the master thesis of the first author [3].

2 Preliminaries

2.1 Graph Classes

Let $x_1, \ldots, x_n \subseteq \mathbb{R}^2$ be a set of n objects, and suppose that there is a distinguished point $p(x_i) \in x_i$, in every object x_i. The *generalized transmission graph* of these objects is a directed graph $G = (V, E)$ with

$$V = \{x_1, \ldots, x_n\} \text{ and } E = \{(x_i, x_j) \mid p(x_j) \in x_i, 1 \leq i, j \leq n\}.$$

We will consider generalized transmission graphs for line segments and circular sectors. In these cases, the distinguished points $p(x_i)$ are defined as follows: for line segments, we choose one fixed endpoint; for circular sectors, we choose the apex.

When constructing arrangements of line segments and of circular sectors below, in Sects. 3 and 4, we need some notation. A line segment ℓ is described by an *endpoint* $p(\ell)$, a *length* $r(\ell)$, and a *direction* $u(\ell)$. A *circular sector* c is presented by an *apex* $p(c)$, a *radius* $r(c)$, an *opening angle* $\alpha(c)$, and a direction $u(c)$. The direction is a vector in $\mathbb{R}^2$, and it indicates the direction of the bisector. We will call the bounding line segments the *outer* line segments of c. Let $B(c)$ be the smallest rectangle with two sides parallel to $u(c)$ that contains c, the *bounding box* of c.

2.2 Stretchability and Combinatorial Descriptions

Let $\mathcal{L}$ be an arrangement of n non-vertical lines, such that no two lines in $\mathcal{L}$ are parallel. We define the *combinatorial description* $D(\mathcal{L})$ of $\mathcal{L}$ as follows:

Let g be a vertical line that lies to the left of all intersection points of $\mathcal{L}$. We number the lines $\ell_1, \dots, \ell_n$ in the order in which they intersect g, from top to bottom. This ordering corresponds to the ascending order of the slopes. For each line ℓ_i, $i = 1, \dots, n$, we have a list O^i of the following form:

$$O^i = (o^i_1, \dots, , o^i_k) \qquad\qquad o^i_j \subseteq \{1, \dots, , n\}$$

$$\bigcup_{j=1}^{k} o^i_j = \{1, ..., n\} \qquad\qquad o^i_j \cap o^i_{j'} = \emptyset, \text{ for } j \neq j'.$$

For $i = 1, \dots, n$, the order of the indices in O^i indicates the order in which the lines ℓ_j cross ℓ_i, as we travel along ℓ_i from left to right. The lists O^i, for $i = 1, \dots, n$, form the *combinatorial description* of the arrangement $\mathcal{L}$. If $\mathcal{L}$ is simple, each o^i_j is a singleton.

Given a combinatorial description $\mathcal{D}$ as above, it is relatively easy to detect whether it comes from an arrangement of *pseudo-lines*. This can be done by checking a few simple axioms [4]. However, the decision problem STRETCHABILITY of deciding if $\mathcal{D}$ originates from an actual arrangement of *lines* turns out to be significantly harder. If all sets o^i_j are singletons, the same problem is called SIMPLE-STRETCHABILITY. Both variants of the problem are complete for $\exists\mathbb{R}$ [5,6].

3 Line Segments

We now present our first result on the recognition of intersection graphs of line segments.

Theorem 3.1. *Recognizing a generalized transmission graph of line segments is* $\exists\mathbb{R}$*-hard.*

Proof. The proof proceeds by a reduction from SIMPLE-STRETCHABILITY. Given an alleged description $\mathcal{D}$ of a simple arrangement of lines, we construct a graph $G_L = (V_L, E_L)$ such that $\mathcal{D}$ is realizable as a line arrangement if and only if G_L is the generalized transmission graph of an arrangement of line segments. We set $V_L = A \cup B \cup C$ with

$$\begin{aligned} A &= \{a_{\{i,k\}} &&\mid 1 \leq i \neq k \leq n\}, \\ B &= \{b^i_k &&\mid 1 \leq i \leq n, 1 \leq k \leq n-1\}, \\ C &= \{c_i &&\mid 1 \leq i \leq n\}, \end{aligned}$$

where the c_i are numbered in order given by $\mathcal{D}$. The { } in the indices of the $a_{\{i,k\}}$ indicates that $a_{\{i,k\}} = a_{\{k,i\}}$.

Before defining the edges, we describe the intuitive meaning of the different vertices. The line segments associated with C correspond to the lines ℓ_i of the arrangement. The endpoints of the line segment associated with $a_{\{i,k\}}$ will enforce that there is an intersection of the line segments for c_i and c_k, for $1 \leq i \neq k \leq n$. The endpoints of the line segments for the b^i_k, $k = 1, \ldots, n-1$, will be placed between the $a_{\{i,k\}}$ on c_i and thus enforce the order of the intersection. When it is clear from the context, we will not explicitly distinguish between a vertex of the graph and the associated line segment. Now we define the edges:

$$\begin{aligned} E_L \;=\; & \{(c_i, a_{\{i,k\}}), (c_i, b^i_k), (b^i_k, c_i) \mid 1 \leq i \neq k \leq n\} \\ & \cup \{(b^i_{o^i_k}, b^i_{o^i_l}), (b^i_{o^i_k}, a_{\{i,o^i_l\}}) \qquad \mid 1 \leq i \leq n, 1 \leq l < k \leq n-1\} \end{aligned}$$

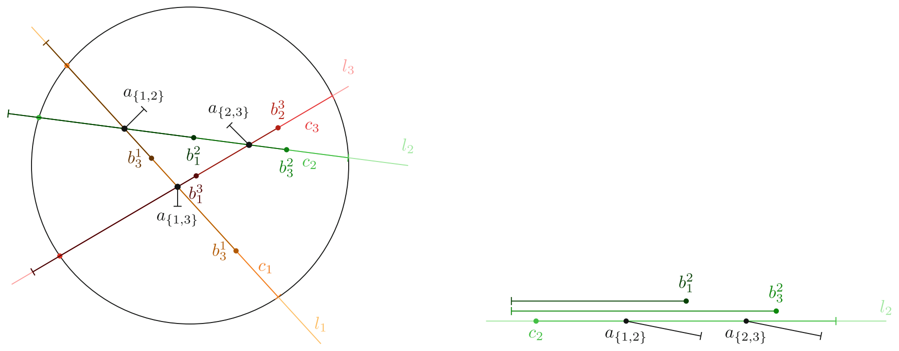
Given $\mathcal{D}$, the sets V_L and E_L can be constructed in polynomial time. It remains to show correctness. Suppose first that $\mathcal{D}$ is realizable, and let $\mathcal{L} = (\ell_1, \ldots, \ell_n)$ be a simple line arrangement with $\mathcal{D} = \mathcal{D}(\mathcal{L})$. We show that there exists an arrangement $\mathcal{C}$ of line segments that realizes G_L. Let D be a disk that contains all vertices of $\mathcal{L}$, with ∂D having a positive distance from each vertex.

The circular order of the intersections between $\ell_1, \ldots, \ell_n$ and ∂D is $\ell_1, \ldots, \ell_n$, $\ell_1, \ldots, \ell_n$. There is no vertical line in $\mathcal{L}$, so we can add a virtual vertical line ℓ' that divides the intersection points along ∂D into a "left" set $D_l = \{q^l_1, q^l_2, \ldots, q^l_n\}$ and a "right" set $D_r = \{q^r_1, q^r_2, \ldots, q^r_n\}$ such that each set contains exactly one intersection with each line ℓ_i, $i = 1, \ldots, n$.

For $i = 1, \ldots, n$, we set c_i to $\ell_i \cap D$, with $p(c_i) = q^l_i$. The $a_{\{i,k\}}$ are constructed such that $p(a_{\{i,k\}})$ is the intersection point of ℓ_i and ℓ_k. The direction and length are chosen in such a way that $a_{\{i,k\}}$ intersects no other lines. Now we place the line segments $b^i_{o^i_k}$. They are positioned such that $p(b^i_{o^i_k})$ lies between $p(a_{\{i,o^i_k\}})$ and $p(a_{\{i,o^i_{k+1}\}})$, for $k = 1, \ldots, n-2$. Furthermore, we place $p(b_{o_{n-1}})$ to the right of $a_{\{i,o^i_{n-1}\}}$. The line segments lie on the lines ℓ_i such that $p(c_i)$ lies in the relative interior of b^i_k. For an example of this construction, see Fig. 1. It follows from the construction that the generalized transmission graph of $\mathcal{C}$ is indeed G_L.

Now consider an arrangement $\mathcal{C}$ of line segments realizing G_L. Let $\mathcal{L}' = (\ell'_1, \ldots, \ell'_n)$ be the arrangement of lines where ℓ'_i is the supporting line of c_i, for $i = 1, \ldots, n$. We claim that $\mathcal{D} = \mathcal{D}(\mathcal{L}')$.

We first consider the role of the line segments $a_{\{i,k\}}$. Since $p(a_{\{i,k\}})$ lies on c_i and c_k, we have $p(a_{\{i,k\}}) = c_i \cap c_k$, and therefore ℓ'_i and ℓ'_k intersect in $p(a_{\{i,k\}})$. This ensures that all pairs of lines have an intersection point that is also the endpoint of an $a_{\{i,k\}}$. Next, we have to show that the order of the intersections along each line ℓ'_i, for $i = 1, \ldots, n$, is in the order as given by $\mathcal{D}$. This is guaranteed by the line segments b^i_k as follows: By the definition of E_L, namely by the edges (c_i, b^i_k) and (b^i_k, c_i), it is ensured that all $p(b^i_k)$ lie on the same line as c_i. The definition also enforces the order of the $p(a_{\{i,k\}})$ and $p(b^i_k)$ along the line. Since $p(a_{\{i,o_k\}})$ lies on $b^i_{o_{k+1}}$ but not on $b^i_{o_k}$ and since all lie on the same line c_i, it has to lie between the corresponding endpoints. This enforces the correct order of the intersections.



(a) Complete line segment construction for three lines

(b) Closeup of c_2. The line segments b_1^2 and b_3^2 are shifted upwards to show their positioning.

Fig. 1. Construction of the line segments.

4 Circular Sectors

We now consider the problem of recognizing generalized transmission graphs of circular sectors. The reduction extends the proof for Theorem 3.1, but we need to be more careful in order to enforce the correct order of intersection.

We will only consider circular sectors with opening angle $\alpha \leq \pi/4$. If x and y are circular sectors with $p(x) \in y$ and $p(y) \in x$, we call x and y a *mutual couple* of circular sectors. We write $\gamma(u(x), u(y))$ for the counter-clockwise angle between the vectors $u(x)$ and $u(y)$.

Observation 4.1. *Let x and y be a mutual couple of circular sectors, then*

$$|\pi - \gamma(u(x), u(y))| \leq (\alpha(x) + \alpha(y))/2.$$

The argument is visualized in Fig. 2a.

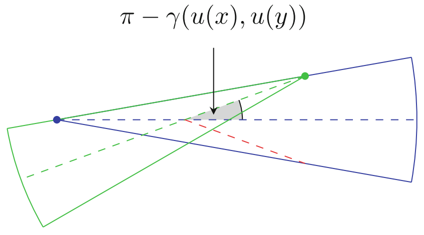
Observation 4.2. *Let x and y be circular sectors whose bisectors intersect at an acute angle of $\beta > \max\{\alpha(x), \alpha(y)\}/2$. Then, the acute angle between the outer line segments of x and the bisector of y is at least $\beta - \max\{\alpha(x), \alpha(y)\}/2$.*

Lemma 4.3. *Let l be a circular sector and let $a_1, \ldots, a_n$ be circular sectors with*

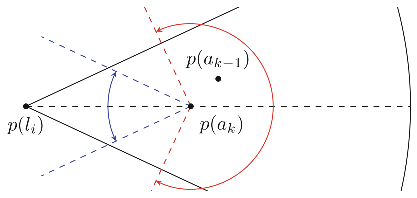
$$\begin{aligned} p(a_i) &\in l, && 1 \leq i \leq n, \\ p(a_i) &\in a_j, && 1 \leq i < j \leq n, \text{ and} \\ p(l) &\in a_j, && 1 \leq j \leq n. \end{aligned}$$

Then, the projection of the $p(a_i)$ onto the directed line ℓ defined by $u(l)$ has the order

$$O = o_1, \ldots, o_n = a_1, \ldots, a_n.$$



(a) Extreme position of x and y; the symmetric case is indicated by the red line.



(b) a_k and l_i form a mutual couple, so $u(a_k)$ lies in the blue range. The apex of a_{k-1} is projected to the right of $p(a_k)$, forcing $u(a_k)$ to be in the red range.

Fig. 2. Illustration of Observation 4.1 and Lemma 4.3

Proof. Each a_i forms a mutual couple with l. Thus, with Observation 4.1, we get

$$|\pi - \gamma\,(u(a_i), u(l))\,| \leq \pi/4. \tag{1}$$

Assume that the order of the projection differs from O. Let $O' = o'_1, \dots, o'_n$ be the actual order of the projection of the $p(a_i)$ onto ℓ. Let j be the first index with $o'_j \neq o_j$ and $o'_j = a_k$. Then, there is an o'_i, $i > j$, with $o'_i = a_{k-1}$. By definition, $p(a_{k-1})$ has to be included in a_k, while still being projected on ℓ to the right of p_k. This is only possible if

$$|\pi - \gamma(u(a_j), \ell)| > \frac{\pi}{2} - \frac{\alpha(a_k)}{2} \geq \frac{\pi}{2} - \frac{\pi}{8} = \frac{3\pi}{8} > \frac{\pi}{4}$$

This is a contradiction to (1), and consequently the order of the projection is as claimed. The possible ranges of the angles are illustrated in Fig. 2b.

An arrangement $\mathcal{C}$ of circular sectors is called *equiangular* if $\alpha(c) = \alpha(c')$ for all circular sectors $c, c' \in \mathcal{C}$.

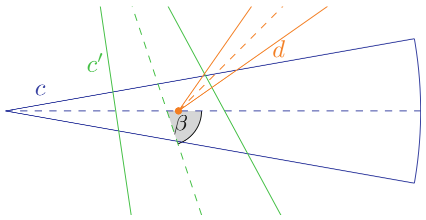
Let c, c' be two circular sectors of $\mathcal{C}$, and assume that $d \in \mathcal{C}$ is a circular sector with $p(d) \in c$ and $p(d) \in c'$, such that c and c' do not form both a mutual couple with the same circular sector. Moreover let $\beta_{\min}$ be the smallest acute angle between the bisector of any pair c, c' with this property. We will call the arrangement *wide spread* if

$$\beta_{\min} \geq 2 \cdot \max_{c \in \mathcal{C}}(\alpha(c))$$

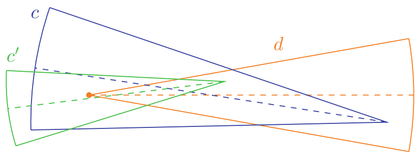
The possible situations are depicted in Fig. 3.

Definition 4.4. *The recognition problem of the generalized transmission graphs of equiangular, wide spread circular sectors is called* SECTOR.

Now we want to show that SECTOR is hard for $\exists\mathbb{R}$. This is done in three steps. First, we give a polynomial-time construction that creates an arrangement of circular sectors from an alleged combinatorial description of a line arrangement. Then we show that this construction is indeed a reduction and therefore show the $\exists\mathbb{R}$-hardness of SECTOR.



(a) Constraint on the angle.



(b) No constraint on the angle.

Fig. 3. The wide spread condition.

Construction 4.5. Given a description $\mathcal{D}$ where all o_i are singletons, we construct a graph $G_L = (V_L, E_L)$. For this construction, let $1 \leq i, k, l \leq n$, $1 \leq m, m', m'' \leq 3$. The set of vertices is defined as follows:

$$V_L = \quad \{c_{im}\} \cup \{a^{im}_{km'} \mid i \neq k\} \cup \{b^{im}_{km'} \mid i \neq k\}$$

As for the line segments, we do not distinguish between the vertices and the circular sectors. For the vertices $a^{im}_{km'}$ and $b^{im}_{km'}$, the upper index indicates the c_{im} with whom $a^{im}_{km'}$ and $b^{im}_{km'}$ form a mutual couple. The lower index hints at a relation to $c_{km'}$. In most cases, the upper index is im and the lower index differs. For better readability, the indices are marked bold ($a^{\mathbf{km'}}_{\mathbf{im}}$), if im is the lower index.

The bisectors of the circular sectors c_{i2} will later define the lines of the arrangement. The circular sectors $a^{im}_{km'}$ and $a^{\mathbf{km'}}_{\mathbf{im}}$ have a similar role as the $a_{\{i,k\}}$ in the construction for the line segments. They enforce the intersection of c_{im} and $c_{km'}$. Similar to the b^i_k, the $b^{im}_{km'}$ help enforcing the intersection order.

We describe E_L on a high level. For a detailed technical description, refer to Appendix A.1. We divide the edges of the graph into *categories*. The first category, E_I, contains the edges that enforce an intersection between two circular sectors c_{im} and $c_{km'}$, for $k < l$. The edges of the next category E_C enforce that each $a^{im}_{km'}$ and each $b^{im}_{km'}$ forms a mutual couple with c_{im}.

$$\begin{aligned} E_I = \quad & \{(c_{im}, a^{im}_{km'}) \mid i \neq k\} \\ & \cup\{(c_{im}, a^{km'}_{im}) \mid i \neq k\} \end{aligned} \qquad \begin{aligned} E_C = \quad & \{(a^{im}_{km'}, c_{im}) \mid i \neq k\} \\ & \cup\{(c_{im}, b^{im}_{km'}) \mid i \neq k\} \\ & \cup\{(b^{im}_{km'}, c_{im}) \mid i \neq k\} \end{aligned}$$

The edges in the next categories enforce the local order. The first category, called E_{GO}, enforces a global order in the sense that the apexes of all $a^{im}_{o_jm'}$ and $b^{im}_{o_jm'}$ will be projected to the left of any $a^{im}_{o_km'}$ and $b^{im}_{o_km'}$ with $k > j$. Additionally, all $a^{\mathbf{o_jm'}}_{\mathbf{im}}$ will be included in $a^{im}_{o_km'}$ and $b^{im}_{o_km'}$. The projection order is enforced by the construction described in Lemma 4.3, the inclusion is enforced by adding the appropriate edges.

It remains to consider the local order of the six circular sectors ($a^{im}_{j1}, \ldots, a^{im}_{j3}$, $b^{im}_{j1}, \ldots, b^{im}_{j3}$) that are associated with c_{im} for each intersecting circular sector

c_{j2}. The projection order of these is either "1, 2, 3" or "3, 2, 1", depending on the order of l_i and l_j on the vertical line. If l_j is below l_i, the order on c_{im} is "1, 2, 3"; in the other case, it is "3, 2, 1". This is again enforced by adding the edges as defined in Lemma 4.3. For a possible realization of this graph, see Figs. 4 and 5. This construction can be carried out in polynomial time.

Now we show that Construction 4.5 gives us indeed a reduction:

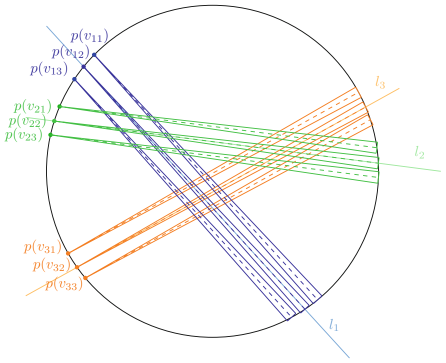


Fig. 4. Construction of the circular sectors c_{im} based on a given line arrangement

Lemma 4.6. *Suppose there is a line arrangement $\mathcal{L} = \{\ell_1, \dots, \ell_n\}$ realizing $\mathcal{D}$, then there is an equiangular, wide spread arrangement $\mathcal{C}$ of circular sectors realizing G_L as defined in Construction 4.5.*

Proof. We construct the containing disk D, and the sets of intersection points D_l and D_r as in the proof of Theorem 3.1. By ℓ_{im}, we denote the directed line through the bisector of the circular sector c_{im}. Let $\alpha_{\min}$ be the smallest acute angle between any two lines of $\mathcal{L}$. The angle α for $\mathcal{C}$ will be set depending on $\alpha_{\min}$ and the placement of the constructed circular sectors c_{im}.

In the first step, we place the circular sectors c_{i2}. They are constructed such that their apexes are on q_i^l and their bisectors are exactly the line segments $\ell_i \cap D$. We place $p(c_{i1})$ in clockwise direction next to $p(c_{i2})$ onto the boundary of D. The distance between $p(c_{i1})$ and $p(c_{i2})$ on ∂D is some small $\tau > 0$. The point $p(c_{i3})$ is placed in the same way, but in counter-clockwise direction from $p(c_{i2})$. The bisectors of all c_{im} are parallel. The radii for c_{i1} and c_{i3} are chosen to be the length of the line segments $\ell_{i1} \cap D$ and $\ell_{i3} \cap D$.

The distance τ must be small enough so that no intersection of any two original lines lies between ℓ_{i1} and ℓ_{i3}. Let β be the largest angle such that if the angle of all c_{im} is set to β, there is always at least one point in c_{im} between the bounding boxes B of two circular sectors with consecutively intersecting bisectors. Since $\mathcal{L}$ is a simple line arrangement, this is always possible. The angle α for the construction is now set to $\min\{\alpha_{\min}/2, \beta\}$. This first part of the construction is illustrated in Fig. 4.

Now we place the remaining circular sectors. Their placement can be seen in Fig. 5. The points $p(a^{im}_{km'})$ all lie on ℓ_{im} with a distance of δ to the left of the intersection of ℓ_{im} and $\ell_{km'}$. By "to the left", we mean that the point lies closer to $p(c_{im})$ on the line ℓ_{im} than the intersection point. The distance δ is chosen small enough such that $p(a^{im}_{km'})$ lies inside of all $a^{\mathbf{km'}}_{\mathbf{im}}$ that have a larger distance to $p(c_{\mathbf{km'}})$ than $p(a^{im}_{km'})$. The direction of the circular sector $a^{im}_{km'}$ is set to $-u(c_{im})$, and its radius is set to $r(a^{im}_{o_k m'}) = \mathrm{dist}(p(a^{im}_{km'}), p(c_{im})) + \varepsilon$, for $\varepsilon > 0$. This lets $p(c_{im})$ lie on the bisecting line segment of every circular sector $a^{im}_{km'}$. The directions and radii for the $b^{im}_{km'}$ are chosen in the same way as for the $a^{im}_{km'}$. The apexes of $b^{im}_{km'}$ are placed such that they lie between the corresponding bounding boxes $B(c_{km'})$. For α small enough, this is always possible.

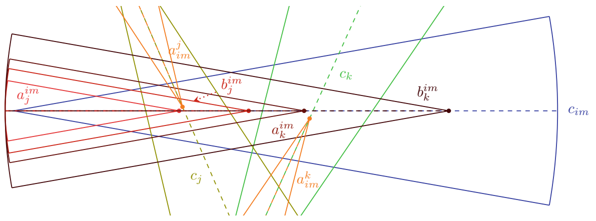


Fig. 5. Detailed construction inside of one circular sector c_{im}.

It follows directly from the construction that the generalized transmission graph of this arrangement is G_L. A detailed argument can be found in Appendix A.2.

Lemma 4.7. *Suppose there is an equiangular, wide spread arrangement $\mathcal{C}$ of circular sectors realizing G_L as defined in Construction 4.5, then there is an arrangement of lines realizing $\mathcal{D}$.*

Proof. From $\mathcal{C}$, we construct an arrangement $\mathcal{L} = (\ell_1, \ldots, \ell_n)$ of lines such that $\mathcal{D}(\mathcal{L}) = \mathcal{D}$ by setting ℓ_i to the line spanned by $u(c_{i2})$. Now, we show that this line arrangement indeed satisfies the description, e.g., that the intersection order of the lines is as indicated by the description.

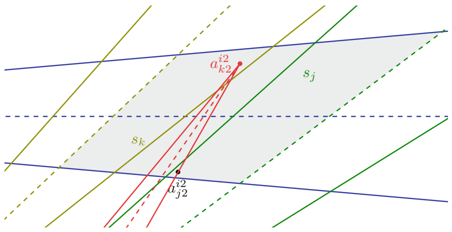
All $a^{im}_{km'}$ and $b^{im}_{km'}$ form mutual couples with c_{im}. Thus, Lemma 4.3 can be applied to them. It follows that the order of the projections of the apexes of the circular sectors is known. In particular, the order of projections of the $p(a^{i2}_{j2})$ onto ℓ_i is the order given by $\mathcal{D}$ and $p(b^{i2}_{o_j 2})$ is projected between $p(a^{i2}_{o_j 2})$ and $p(a^{i2}_{o_{j+1} 2})$.

Now, we have to show that the order of intersections of the lines corresponds to the order of the projections of the $p(a^{i2}_{j2})$. This will be done through a contradiction. We consider two circular sectors c_{j2} and c_{k2}. Assume that the order of the projection of the apexes of a^{i2}_{j2} and a^{i2}_{k2} onto ℓ_i is $p(a^{i2}_{j2})$, $p(a^{i2}_{k2})$, while the order of intersection of the lines is ℓ_k, ℓ_j.

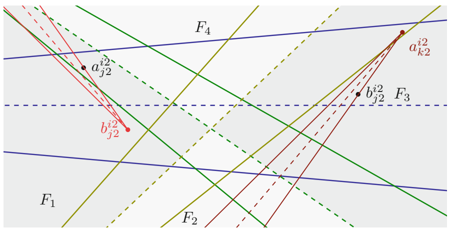
Note that by the definition of the edges of G_L, c_{j2} and c_{k2} share the apexes of a_{j2}^{k2} and a_{k2}^{j2}, but there is no circular sector they both form a mutual couple with and thus the angle between their bisecting line segments is large.

There are two main cases to consider, based on the position of the intersection point p of ℓ_j and ℓ_k relative to c_{i2}:

Case one $p \notin c_{i2}$: If p does not lie in c_{i2}, then ℓ_j and ℓ_k divide c_{i2} into three parts. Let s_j, s_k be the outer line segments of c_{j2} and c_{k2} that lie in the middle part of this decomposition. A schematic of this situation can be seen in Fig. 6a.



(a) Case one, a_{k2}^{i2} cannot reach a_{j2}^{i2}.



(b) Case two, b_{j2}^{i2} cannot lie in F_1 or F_3.

Fig. 6. Position of p relative to c_{i2}

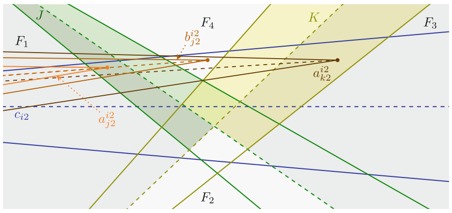
From Observation 4.2 and since $\mathcal{C}$ is an equiangular, wide spread arrangement it follows that $|\pi - \gamma(s_j, u(c_i))| > 3\alpha/2$ and $|\pi - \gamma(s_k, u(c_i))| > 3\alpha/2$.

In order to have an intersection order that differs from the projection order, the circular sector a_{k2}^{i2} has to reach $p(a_{j2}^{i2})$. The latter point is projected to the left of a_{k2}^{i2} but lies right of s_k. The directed line segment d from $p(a_{k2}^{i2})$ to $p(a_{k2}^{j2})$ has to intersect s_j and s_k, and thus it has to hold that $|\pi - \gamma(d, u(c_{i2}))| \geq 3\alpha/2$. The line segment d has to lie inside of a_{k2}^{i2}, which is only possible if $|\pi - \gamma(u(a_{k2}^{i2}), u(c_i))| > \alpha$. However, this is a contradiction to $|\pi - \gamma(u(a_k^i), u(c_i))| \leq \alpha$, which follows from Observation 4.1.

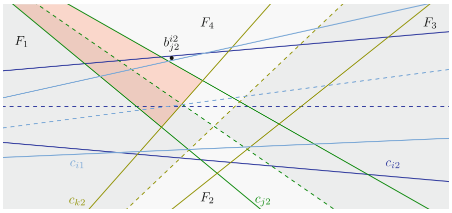
Case two $p \in c_{i2}$: W.l.o.g., let $u(c_{i2}) = \lambda \cdot (1, 0)$, $\lambda > 0$, and let $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$ be the decomposition of the plane into faces induced by ℓ_j and ℓ_k. Here, F_1 is the face with $p(c_{i2})$, and the faces are numbered in counter-clockwise order.

We consider the possible placements of $p(b_{j2}^{i2})$ in one of the face. First, we show that $p(b_{j2}^{i2})$ cannot lie in F_1 or in F_3. From the form of E_{GO}, we know that $p(a_{j2}^{i2})$ has to be projected left of $p(b_{j2}^{i2})$ and $p(a_{j2}^{i2})$ has to lie inside of b_{j2}^{i2}; see Fig. 6b for a schematic of the situation. If $p(b_{j2}^{i2})$ lies in F_1, the line segment in b_{j2}^{i2} that connects $p(b_{j2}^{i2})$ and $p(a_{j2}^{i2})$ has to cross an outer line segment of c_{j2}. This yields the same contradiction as in the first case. If $p(b_{j2}^{i2})$ were in F_3, an analogous argument holds for $p(b_{j2}^{i2})$, which has to lie inside of a_{k2}^{i2}.

This leaves F_2 and F_4 as possible positions for b_{j2}^{i2}. W.l.o.g., let b_{j2}^{i2} be located in F_4. We divide c_{j2} and c_{k2} by ℓ_k or ℓ_j, respectively, into two parts, and denote the parts containing the line segments that are incident to F_4 by J and K. Then, again by using that the arrangement is wide spread, it can be seen that $p(a_{j2}^{i2})$ and $p(a_{k2}^{i2})$ are located in J and K. The possible placement is visualized in Fig. 7a.



(a) The localization of a_{j2}^{i2} and a_{k2}^{i2}.



(b) p can not lie in c_{i1}.

Fig. 7. Case two and relative order

The argument so far yields that if $p \in c_{i2}$, then the intersection order of ℓ_j and ℓ_k with ℓ_i is the same as the order of projection if ℓ_i lies above p, and is the inverse order if ℓ_i lies below p. The uncertainty of this situation is not desirable. By considering the circular sectors c_{i1} and c_{i3}, we will now show that such a situation cannot occur.

First, we show that c_{i1} and c_{i3} cannot contain the intersection point of ℓ_j and ℓ_k. W.l.o.g., assume that the intersection point lies in c_{i1}. Then, b_{j2}^{i1} is included in either F_2 or F_4. Consider the case that b_{j2}^{i1} lies in F_4. Since $u(c_{i2}) = \lambda \cdot (1,0)$ and since one of the outer line segments of c_{i2} has to lie beneath p, there is only one outer line segment of c_{i2} that intersects $F_4 \setminus (J \cup K)$, J and K. There are at most two intersection points of this outer line segment with ∂c_{i1}. This implies that there is no intersection point of ∂c_{i2} and ∂c_{i1} in at least one of J, K, and $F_4 \setminus (J \cup K)$. If there is no intersection point, then c_{i1} and c_{i2} overlap in this interval. W.l.o.g., let this area be J, and let $c_{i1} \cap J$ be fully contained in $c_{i2} \cap J$. Then, $p(a_{j1}^{im})$ cannot be placed. Consequently, this situation is not possible. The argument is depicted in Fig. 7b.

If $p(b_{j2}^{i1})$ was included in F_2, then the order of projection of $p(a_{k2}^{i2})$ and $p(a_{j2}^{i2})$ would be the same order as the order of intersections of ℓ_j and ℓ_k with a parallel line to ℓ_i that lies below ℓ_i. This order is the inverse order of the order of projection in c_{i2}. Since the order of the projection as defined by E_{GO} depends only on k and i, the order of projection of $p(a_{j2}^{im})$ and $p(a_{k2}^{im})$ has to be the same in all c_{im}. This implies that $p(b_{k2}^{i1})$ is not included in F_2.

Now, we know that c_{i1} and c_{i3} do not contain the intersection point. This implies that the argument from the case $p \notin c_{i2}$ can be applied to them and the order of intersection in c_{i1} and c_{i3} is the same as the order of the projections of $p(a_{j2}^{i1})$ and $p(a_{k2}^{i1})$. This order is the same in all three c_{im}, and thus the bisectors of c_{i1} and c_{i3} have to lie on the same side of the intersection point. Furthermore, the points $p(a_{j2}^{i1})$ and $p(a_{j2}^{i3})$ have to lie in J but outside of c_{i2}. This implies that ℓ_{i1} and ℓ_{i3} both intersect ℓ_j and ℓ_k either before ℓ_i or after ℓ_i, while $p(b_{j2}^{i2})$ lies in F_4.

The edges for the local order define that the order of projection onto ℓ_j is $p(a_{j2}^{i1})$, $p(a_{j2}^{i2})$, $p(a_{j2}^{i3})$ (or the reverse), and the analogous statement holds for ℓ_k. This order is not possible with c_{i1} and c_{i3}, both lying above or below c_{i2}, which implies that the intersection point cannot lie in c_{i2}. Since the order of intersection is the same as the order of the projection, if $p \notin c_{i2}$ and a situation with $p \in c_{i2}$ is not possible, we have shown that $\mathcal{D}(\mathcal{L}) = \mathcal{D}$.

With the tools from above, we can now give the proof of the main result of this section:

Theorem 4.8. Sector *is hard for* $\exists\mathbb{R}$.

Proof. The theorem follows from Construction 4.5 and Lemmas 4.6 and 4.7.

5 Conclusion

We have defined the new graph class of generalized transmission graphs as a model for directed antennas with arbitrary shapes. We showed that the recognition of generalized transmission graphs of line segments and a special form of circular sectors is $\exists\mathbb{R}$-hard.

For the case of circular sectors, we needed to impose certain conditions on the underlying arrangements. The wide spread condition in particular seems to be rather restrictive. We assume that this condition can be weakened, if not dropped, while the problem remains $\exists\mathbb{R}$-hard.

Ours are the first $\exists\mathbb{R}$-hardness results on directed graphs that we are aware of. We believe that this work could serve as a starting point for a broader investigation into the recognition problem for geometrically defined directed graph models, and to understand further what makes these problems hard.

Acknowledgments. We would like to thank an anonymous reviewer for pointing out a mistake in Observation 4.1.

A Missing Proofs and Constructions

A.1 Full Construction for SECTOR

Let the vertices of the construction be defined as in Construction 4.5. We divide the edges of the graph into *categories*. The first category E_I contains the edges that enforce an intersection of two circular sectors c_{im} and $c_{km'}$ for $k < l$.

$$E_I = \left\{ \left(c_{im}, a_{km'}^{im}\right), \left(c_{im}, a_{im}^{km'}\right) \,\middle|\, i \neq k \right\}.$$

The edges E_C enforce that each $a^{im}_{km'}$ and each $b^{im}_{km'}$ forms a mutual couple with c_{im}.

$$E_C = \Big\{ \big(a^{im}_{km'}, c_{im}\big), \big(c_{im}, b^{im}_{km'}\big), \big(b^{im}_{km'}, c_{im}\big) \,\Big|\, i \neq k \Big\}.$$

The edges of E_{GO} will enforce the order of the projection of the apexes of $a^{im}_{o_k m'}$, $a^{im}_{o_l m''}$, $b^{im}_{o_k m'}$, and $b^{im}_{o_l m''}$ for $k > l$ onto the bisector of c_{im}. They are chosen such that $p(a^{im}_{o_k m'})$ will be projected closer to $p(c_{im})$ than $p(a^{im}_{o_l m''})$, for $k < l$. Also included in E_{GO} are edges that enforce that all $p(a^{\mathbf{o_k m'}}_{\mathbf{im}})$ are included in the circular sectors $a^{im}_{o_l m''}$ and $b^{im}_{o_l m''}$.

$$\begin{aligned} E_{\text{GO}} \;=\; & \Big\{ (a^{im}_{o_k m'}, a^{im}_{o_l m''}), (a^{im}_{o_k m'}, a^{\mathbf{o_l m''}}_{\mathbf{im}}), (a^{im}_{o_k m'}, b^{im}_{o_l m''}), \\ & (b^{im}_{o_k m'}, a^{im}_{o_l m''}), (b^{im}_{o_k m'}, a^{\mathbf{o_l m''}}_{\mathbf{im}}) \quad \Big|\, i \neq k, k > l \Big\}. \end{aligned}$$

The last two categories of edges will enforce the projection order of the apexes of $a^{im}_{o_k 1}$, $a^{im}_{o_k 2}$, $a^{im}_{o_k 3}$, and $b^{im}_{o_k 1}$, $b^{im}_{o_k 2}$, $b^{im}_{o_k 3}$ onto the bisector of c_{im}. This order is $a^{im}_{o_k 1}$, $b^{im}_{o_k 1}$, $a^{im}_{o_k 2}$, $b^{im}_{o_k 2}$, $a^{im}_{o_k 3}$ $b^{im}_{o_k 3}$, if $o_k > i$, and the inverse order, otherwise. The edges for the first case are E_{LOI}, and the edges for the second case are E_{LOD}. We set

$$\begin{aligned} E_{\text{LOI}} \;=\; & \Big\{ (a^{im}_{o_k m'}, a^{im}_{o_k m''}), (a^{im}_{o_k m'}, a^{\mathbf{o_k m''}}_{\mathbf{im}}), \\ & (a^{im}_{o_k m'}, b^{im}_{o_k m''}), (b^{im}_{o_k m'}, b^{im}_{o_k m''}) \;\Big|\, i \neq k, m'' < m', o_k > i \Big\} \\ & \cup \Big\{ (b^{im}_{o_k m'}, a^{im}_{o_k m''}), (b^{im}_{o_k m'}, a^{\mathbf{o_k m''}}_{\mathbf{im}}) \;\Big|\, i \neq k, m'' \leq m', o_k > i \Big\} \end{aligned}$$

and

$$\begin{aligned} E_{\text{LOD}} \;=\; & \Big\{ (a^{im}_{o_k m'}, a^{im}_{o_k m''}), (a^{im}_{o_k m'}, a^{\mathbf{o_k m''}}_{\mathbf{im}}), \\ & (a^{im}_{o_k m'}, b^{im}_{o_k m''}), (b^{im}_{o_k m'}, b^{im}_{o_k m''}) \;\Big|\, i \neq k, m'' > m', o_k < i \Big\} \\ & \cup \Big\{ (b^{im}_{o_k m'}, a^{im}_{o_k m''}), (b^{im}_{o_k m'}, a^{\mathbf{o_k m''}}_{\mathbf{im}}) \;\Big|\, i \neq k, m'' \geq m', o_k < i \Big\}. \end{aligned}$$

The set of all edges is defined as

$$E_L = E_I \cup E_C \cup E_{\text{GO}} \cup E_{\text{LOI}} \cup E_{\text{LOD}}.$$

A.2 Remaining Proof for Lemma 4.6

Lemma A.1. *The generalized transmission graph of the arrangement $\mathcal{C}$ of circular sectors constructed in Lemma 4.6 is G_L.*

Proof. As δ is chosen small enough that $a^{im}_{km'}$ and $a^{\mathbf{km'}}_{\mathbf{im}}$ lie in c_{im}, the edges of E_I are created. Since $b^{im}_{km'}$ and $a^{im}_{km'}$ have the inverse direction of c_{im} and the radii are large enough, $p(c_{im})$ is included in $a^{im}_{km'}$ and in $b^{im}_{km'}$. Hence all edges in E_C are created.

By the choice of the radii and the direction, $a^{im}_{o_k m'}$ includes all apexes of circular sectors that lie on ℓ_{im} and closer to $p(c_m)$ than $p(a^{im}_{o_k m'})$. Furthermore, δ is small enough such that all $a^{\mathbf{o_l m}''}_{\mathbf{im}}$, $l < k$, are included in $a^{im}_{o_k m'}$. This implies that edges from E_{GO} are present in the generalized transmission graph of $\mathcal{C}$.

The only edges that have not been considered yet are the edges in E_{LOI} and E_{LOD}. For a circular sector $a^{im}_{o_k m'}$ with $o_k > i$, the slope of ℓ_{o_k} is larger than the slope of ℓ_i. By the counter-clockwise construction, $\ell_{o_k 1}$ lies above $\ell_{o_k 2}$. This implies that the intersection point of $\ell_{o_k 1}$ and ℓ_{im} lies closer to $p(c_{im})$ than the intersection points with $\ell_{o_k 2}$ or $\ell_{o_k 3}$. The presence of the edges can now be seen by the same argument as for the edges of E_{GO}. Symmetrical considerations can be made for the edges of E_{LOD}.

It remains to show that no additional edges are created. Note that all apexes of the circular sectors lie inside of D and that all $a^{im}_{km'} \cap D$ and $b^{im}_{km'} \cap D$ are included in the boxes $B(c_{im})$.

Since only the apexes of $a^{im}_{km'}$, $a^{\mathbf{km}'}_{\mathbf{im}}$, and $b^{im}_{km'}$ lie in c_{im}, there are no additional edges starting at c_{im}. The rectangles $B(c_{im})$ are disjoint on the boundary of D and all $a^{im}_{km'} \cap D$ and $b^{im}_{km'} \cap D$ lie inside of $B(c_{im})$. This implies that there are no additional edges ending at c_{im}. Now, we have to consider additional edges starting at $a^{im}_{km'}$ and $b^{im}_{km'}$. Note that $\alpha \leq \pi/4$ enforces that no circular sector $a^{im}_{km'}$ or $b^{im}_{km'}$ can reach an apex having a larger distance to $p(c_{im})$. Also, note that there are edges for all circular sectors with smaller distances in E_{GO}, E_{LOD} or E_{LOI}. This covers all possible additional edges.

References

1. Kang, R.J., Müller, T.: Sphere and dot product representations of graphs. Discret. Comput. Geom. **47**(3), 548–568 (2012)
2. Kaplan, H., Mulzer, W., Roditty, L., Seiferth, P.: Spanners and reachability Oracles for directed transmission graphs. In: Proceedings of the 34th International Symposium on Computational Geometry (SoCG), pp. 156–170 (2015)
3. Klost, K.: Complexity of recognizing generalized transmission graphs, March 2017. http://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Klost17.pdf
4. Knuth, D.E.: Axioms and Hulls. LNCS, vol. 606. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55611-7
5. Matoušek, J.: Intersection graphs of segments and $\exists\mathbb{R}$ (2014). arXiv:1406.2636
6. Mnëv, N.E.: Realizability of combinatorial types of convex polyhedra over fields. J. Sov. Math. **28**(4), 606–609 (1985)
7. Schaefer, M.: Complexity of some geometric and topological problems. In: Proceeding of the 17th International Symposium on Graph Drawing (GD), pp. 334–344 (2009)

A Tight Lower Bound for an Online Hypercube Packing Problem and Bounds for Prices of Anarchy of a Related Game

Yoshiharu Kohayakawa[1], Flávio Keidi Miyazawa[2], and Yoshiko Wakabayashi[1](✉)

[1] Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil
{yoshi,yw}@ime.usp.br

[2] Institute of Computing, University of Campinas, Campinas, Brazil
fkm@ic.unicamp.br

Abstract. We prove a tight lower bound on the asymptotic performance ratio ρ of the *bounded space online d-hypercube bin packing problem*, solving an open question raised in 2005. In the classic d-hypercube bin packing problem, we are given a sequence of d-dimensional hypercubes and we have an unlimited number of bins, each of which is a d-dimensional unit hypercube. The goal is to pack (orthogonally) the given hypercubes into the minimum possible number of bins, in such a way that no two hypercubes in the same bin overlap. The *bounded space online d-hypercube bin packing problem* is a variant of the d-hypercube bin packing problem, in which the hypercubes arrive *online* and each one must be packed in an open bin without the knowledge of the next hypercubes. Moreover, at each moment, only a constant number of open bins are allowed (whenever a new bin is used, it is considered open, and it remains so until it is considered closed, in which case, it is not allowed to accept new hypercubes). Epstein and van Stee (SIAM J Comput 35(2):431–448, 2005) showed that ρ is $\Omega(\log d)$ and $O(d/\log d)$, and conjectured that it is $\Theta(\log d)$. We show that ρ is in fact $\Theta(d/\log d)$. To obtain this result, we elaborate on some ideas presented by those authors, and go one step further showing how to obtain better (offline) packings of certain special instances for which one knows how many bins any bounded space algorithm has to use. Our main contribution establishes the existence of such packings, for large enough d, using probabilistic arguments. Such packings also lead to lower bounds for the prices of anarchy of the selfish d-hypercube bin packing game. We present a lower bound of $\Omega(d/\log d)$ for the pure price of anarchy of this game, and we also give a lower bound of $\Omega(\log d)$ for its strong price of anarchy.

The first author was partially supported by FAPESP (2013/03447-6, 2013/07699-0), CNPq (310974/2013-5, 459335/2014-6) and NUMEC/USP (Project MaCLinC). The second author was partially supported by CNPq (311499/2014-7, 425340/2016-3) and FAPESP (2015/11937-9, 2016/01860-1, 2016/23552-7). The third author was partially supported by CNPq (456792/2014-7, 306464/2016-0) and FAPESP (2015/11937-9).

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 697–711, 2018.
https://doi.org/10.1007/978-3-319-77404-6_51

1 Introduction and Main Results

The bin packing problem is an iconic problem in combinatorial optimization that has been largely investigated from many different viewpoints. In special, it has served as a proving ground for new approaches to the analysis of approximation algorithms. It is one of the first problems for which approximation algorithms were proposed in the beginning of seventies, and also ideas to prove lower bounds for online algorithms and probabilistic analysis first appeared [5]. We believe that the technique we present in this paper is novel and contributes with new ideas that may possibly be incorporated into this area of research.

We prove bounds for two variants of the bin packing problem, in which the items to be packed are d-dimensional cubes (also referred to as *d-hypercubes* or simply hypercubes, when the dimension is clear). More precisely, we show results for the *online bounded space d-hypercube bin packing problem* and the *selfish hypercube bin packing game*. Before we state our results in the next section, we define these problems and mention some known results.

The *d-hypercube bin packing problem* (d-CPP) is defined as follows. We are given a list L of items, where each item $h \in L$ is a d-hypercube of side length $s(h) \leq 1$, and an unlimited number of bins, each of which is a unit d-hypercube. The goal is to find a packing $\mathcal{P}$ of the items of L into a minimum number of bins. More precisely, we have to assign each item h to a bin, and specify its position $(x_1(h), \ldots, x_d(h))$ in this bin. As usual, we consider that each bin is defined by the region $[0,1]^d$, and thus, we must have $0 \leq x_i(h) \leq 1 - s(h)$, for $i = 1, \ldots, d$. Additionally, we must place the items parallel to the axes of the bin and guarantee that items in the same bin do not overlap. The *size* of the packing $\mathcal{P}$ is the number of *used* bins (those with at least one item assigned to it). Throughout this paper, the bins are always assumed to be unit hypercubes of the same dimension of the items that have to be packed.

The d-CPP is in fact a special case of the *d-dimensional bin packing problem* (d-BPP), in which one has to pack d-dimensional parallelepipeds into d-dimensional unit bins. For $d = 1$, both problems reduce to the well known *bin packing problem*.

In the *online* variant of d-CPP, the hypercubes arrive online and must be packed in an open bin (without the knowledge of the next hypercubes). The *online bounded space* variant of the d-CPP is a more restricted variant of the online d-CPP. Whenever a new empty bin is used, it is considered an open bin and it remains so until it is considered closed, after which it is not allowed to accept other hypercubes. In this variant, during the packing process, only a constant number of open bins is allowed. The corresponding problem or algorithm in which the whole list of items is known beforehand is called offline.

As it is usual, for bin packing problems, we consider the asymptotic performance ratio to measure the quality of the algorithms. For an algorithm $\mathcal{A}$, and an input list L, let $A(L)$ be the number of bins used by the solution produced by algorithm A for the list L, and let $\mathrm{OPT}(L)$ be the minimum number of bins needed to pack L. The *asymptotic performance ratio* of algorithm A is defined as

$$\mathcal{R}_A^\infty = \limsup_{n\to\infty} \sup_L \left\{ \frac{A(L)}{\mathrm{OPT}(L)} : \mathrm{OPT}(L) = n \right\}. \tag{1}$$

Given a packing problem Π, the *optimal asymptotic performance ratio* for Π is defined as

$$\mathcal{R}_\Pi^\infty = \inf \{\mathcal{R}_A^\infty : A \text{ is an algorithm for } \Pi\}. \tag{2}$$

Many results have been obtained for the online d-BPP and d-CPP problems (see [2,3,15,16,22,23]). Owing to space limitation, we mention only results for the online bounded space versions of these problems. For the online bounded space 1-BPP, Csirik [6] presented an algorithm with asymptotic performance ratio at most $\Pi_\infty \approx 1.69103$, shown to be an optimal online bounded space algorithm by Seiden [21]. For the online bounded space d-BPP, $d \geq 2$, a lower bound of $(\Pi_\infty)^d$ follows from [7]; Epstein and van Stee [12] showed that this bound is tight.

For the online bounded space d-CPP, Epstein and van Stee [12] showed that its asymptotic performance ratio is $\Omega(\log d)$ and $O(d/\log d)$, and conjectured that it is $\Theta(\log d)$. They also showed an optimal algorithm for this problem, but left as an interesting open problem to determine its asymptotic performance ratio. One of our main results builds upon their work and shows a lower bound that matches the known upper bound.

Theorem 1. *The asymptotic performance ratio of the online bounded space d-hypercube bin packing problem is $\Omega(d/\log d)$.*

In view of the previous results [12], we have that the asymptotic performance ratio of the online bounded space d-hypercube bin packing problem is $\Theta(d/\log d)$. Results on lower and upper bounds for $d \in \{2, \ldots, 7\}$ have also been obtained by Epstein and van Stee [10].

The technique that we use to prove the above theorem can also be used to obtain lower bounds for a game theoretic version of the d-CPP problem, called *selfish d-hypercube bin packing game*.

This game starts with a set of d-hypercubes arbitrarily packed into unit bins. Each of these hypercubes is (controlled by) a player. For simplicity, in the game context, we will use the terms hypercube, item and player in an interchangeable manner. For a game with n items, a *configuration* is a vector $p = (p_1, \ldots, p_n)$, where p_i indicates in which bin item i is packed. (Equivalently, a configuration is a packing of the items into bins.) The *cost* of an item is defined as the ratio between its volume and the total occupied volume of the respective bin. In this game, an item can migrate to another bin only when its cost decreases. Players may act selfishly by changing their strategy (that is, moving to another bin) to minimize their costs. For a given game configuration p, its *social cost*, denoted by $\mathrm{SC}(p)$, is the total cost paid by the players (which is precisely the number of used bins). The *optimal social goal* is a game configuration of minimum social cost, which we denote by $\mathrm{OPT}(L)$.

An important concept in game theory is the Nash equilibrium [20]. In the selfish hypercube bin packing game, a *(pure) Nash equilibrium* is a stable packing where no player can reduce his cost by unilaterally changing his strategy (that is, moving to another bin), while the strategies of all other players remain unchanged. The pure Nash equilibrium may not be resilient to the action of coalitions, as it does not assume that players negotiate and cooperate with each other. Aumann [1] introduced the concept of *strong Nash equilibrium* in coalitional game theory; in this case, a group of players may agree to coordinate their actions in a mutually beneficial way. A *strong Nash equilibrium* is a game configuration where no group of players can reduce the cost of each of its members by changing strategies together, while non-members maintain their strategies.

Throughout the paper, the Nash equilibrium is considered only in the setting of pure strategies (for pure strategies, a player chooses only one strategy at a time, while for mixed strategies, a player chooses an assignment of probabilities to each pure strategy). Given a game G, we denote by $\mathcal{N}(G)$ (resp. $\mathcal{SN}(G)$) the set of configurations in Nash equilibrium (resp. strong Nash equilibrium).

To measure the quality of an equilibrium, Koutsoupias and Papadimitriou [18] proposed a measure in a game-theoretic framework that nowadays is known as the *price of anarchy* (resp. *strong price of anarchy*), which is the ratio between the worst social cost of a Nash equilibrium (resp. strong Nash equilibrium) and the optimal social cost. The price of anarchy measures the loss of the overall performance due to the decentralized environment and the selfish behavior of the players. As it is common for bin packing problems, for bin packing games one also considers asymptotic price of anarchy. The *(asymptotic) price of anarchy* of a class $\mathcal{G}$ of games is defined as

$$\mathrm{PoA}(\mathcal{G}) := \limsup_{m\to\infty} \sup_{G\in\mathcal{G},\, \mathrm{OPT}(G)=m} \max_{p\in\mathcal{N}(G)} \frac{\mathrm{SC}(p)}{m}. \tag{3}$$

The *(asymptotic) strong price of anarchy* of a class $\mathcal{G}$ of games, denoted $\mathrm{SPoA}(\mathcal{G})$, is defined analogously, considering only configurations that are strong Nash equilibria.

We are interested in the case $\mathcal{G}$ is the class of the *selfish d-hypercube bin packing games*, with the natural cost function (proportional model) we have defined. (Note that, other cost functions can also be defined for bin packing games.) We will prove bounds for the asymptotic prices of anarchy of this class of games. The corresponding measures will be denoted by $\mathrm{PoA}(d)$ and $\mathrm{SPoA}(d)$, where d indicates the dimension of the items in the game. Although we may not mention explicitly, the prices of anarchy considered are always asymptotic.

The case $d = 1$ of this game was first investigated by Bilò [4], who referred to it as selfish bin packing game. He proved that this game always converges to a pure Nash equilibrium and proved that $\mathrm{PoA}(1) \in [1.6, 1.666]$. Yu and Zhang [24] improved this result to $\mathrm{PoA}(1) \in [1.6416, 1.6575]$. Epstein and Kleiman [9] obtained (independently) the same lower bound and improved the upper bound to 1.6428; they also proved that $\mathrm{SPoA}(1) \in [1.6067, 1.6210]$. Very recently, Epstein et al. [11] showed that $\mathrm{SPoA}(1) \approx 1.6067$. For $d = 1$, Ma et al. [19], obtained results considering another cost function. The case $d = 2$ was first investigated by Fernandes et al. [13]. They showed in [14] that

PoA(2) $\in$ [2.3634, 2.6875] and SPoA(2) $\in$ [2.0747, 2.3605]. For a survey on bin packing games with selfish items, we refer the reader to Epstein [8].

Our second set of results concern lower and upper bounds for PoA(d) and SPoA(d).

Theorem 2. *Let* PoA(d) *be the price of anarchy of the selfish d-hypercube bin packing game. There is an absolute constant d_0 such that, for all $d \geq d_0$, we have*

$$\mathrm{PoA}(d) \geq \frac{d}{5 \log d}. \tag{4}$$

We remark that our proof of Theorem 2, presented in Sect. 3, may be adapted to prove the following statement: *for any $\varepsilon > 0$ there is $d_0 = d_0(\varepsilon)$ such that, for any $d \geq d_0$, we have* $\mathrm{PoA}(d) \geq (1/4 - \varepsilon)d/\log d$.

Theorem 3. *Let* SPoA(d) *be the strong price of anarchy of the selfish d-hypercube bin packing game. There is an absolute constant d_0 such that, for all $d \geq d_0$, we have*

$$\mathrm{SPoA}(d) \geq \log d. \tag{5}$$

The proof of Theorem 3 uses arguments similar to those used in the proof of Theorem 2 and is therefore omitted. We also prove that the price of anarchy of the selfish d-hypercube bin packing game is at most 2^d. (For the proofs omitted here the reader may refer to [17]). We believe the probabilistic technique used to obtain the lower bounds in Theorems 1, 2 and 3 is novel and promising for obtaining lower bounds for other packing problems and games.

2 Notation, Special Packings and Central Lemmas

The open d-hypercubes $Q_k^d(\varepsilon)$ defined below will be crucial in what follows.

Definition 1. *Let $d \geq 2$ be an integer. For all integer $k \geq 2$ and $0 < \varepsilon \leq 1$, let*

$$Q_k^d(\varepsilon) = (1+\varepsilon)\left(0, \frac{1}{k}\right)^d = \left(0, \frac{1+\varepsilon}{k}\right)^d = \left\{x \in \mathbb{R} : 0 < x < \frac{1+\varepsilon}{k}\right\}^d \subset [0,1]^d \tag{6}$$

be the open d-hypercube of side length $(1+\varepsilon)/k$ 'based' at the origin.

For convenience, given $\varepsilon > 0$ and a positive integer q, we write $q_{-\varepsilon}$ for $q/(1+\varepsilon)$. The quantity ε will often be clear from the context, and in those cases we simply write q_- for $q_{-\varepsilon}$. Note that, for instance, we have

$$Q_k^d(\varepsilon) = \left(0, \frac{1}{k_-}\right)^d. \tag{7}$$

In what follows, we are interested in certain types of packings $\mathcal{U}$ of hypercubes into a unit bin. If a packing $\mathcal{P}$ of hypercubes is made up of packings $\mathcal{U}_1, \ldots, \mathcal{U}_N$, with each $\mathcal{U}_i$ being a packing into a unit bin, then we write $\mathcal{P} = (\mathcal{U}_1, \ldots, \mathcal{U}_N)$, and denote by $|\mathcal{P}|$ the number of bins N in $\mathcal{P}$.

Definition 2 (Packings of type $\mathcal{H}_k^d(\varepsilon)$ and $\mathcal{H}^d(\varepsilon)$). *Let $d \geq 2$ be fixed. For any integer $k \geq 2$ and $0 < \varepsilon \leq 1/(k-1)$, a packing $\mathcal{U}$ of $(k-1)^d$ copies of $Q_k^d(\varepsilon)$ into a unit bin is said to be a packing of type $\mathcal{H}_k^d(\varepsilon)$. A packing $\mathcal{P} = (\mathcal{U}_1, \mathcal{U}_2, \dots)$ is said to be of type $\mathcal{H}^d(\varepsilon)$ if for each i there is some k such that $\mathcal{U}_i$ is a packing of type $\mathcal{H}_k^d(\varepsilon)$.*

In the definition above, the upper bound on ε guarantees that $(k-1)^d$ copies of $Q_k^d(\varepsilon)$ *can* be packed into a unit bin (and hence $\mathcal{H}_k^d(\varepsilon)$ exists): it suffices to notice that, under that assumption on ε, we have $(k-1)(1+\varepsilon)/k \leq 1$.

Packings of type $\mathcal{H}_k^d(\varepsilon)$ and $\mathcal{H}^d(\varepsilon)$ are called *homogeneous packings*. They will be important for us because they are Nash equilibria (see Lemma 3), and also because they can be used to create instances for which any bounded space algorithm performs badly (following ideas of Epstein and van Stee [10,12]).

Two packing lemmas. For the next definition, suppose $\mathbb{D}$ is a given set of integers, and ε is a positive real number.

Definition 3 (Packings of type $(1+\varepsilon)\mathbb{D}^{-1}$). *A packing $\mathcal{U}$ of d-hypercubes into a unit bin is of type $(1+\varepsilon)\mathbb{D}^{-1}$ if, for every member Q of $\mathcal{U}$, there is some integer $k \in \mathbb{D}$ such that Q is a copy of $Q_k^d(\varepsilon)$.*

In what follows, we shall restrict to packings $\mathcal{U}$ of type $(1+\varepsilon)\mathbb{D}^{-1}$, where $\mathbb{D}$ is one of the following sets: (a) $\mathbb{D} = \mathbb{Z}_{\geq 2} = \{k \in \mathbb{Z} \colon k \geq 2\}$ or (b) $\mathbb{D} = \mathbb{Z}_{2^+}$, where $\mathbb{Z}_{2^+}$ denotes the set $\{2^i \colon i \geq 1\}$. Following [9,11], we consider $\mathbb{D} = \mathbb{Z}_{2^+}$ to deal with strong Nash equilibria.

Let $\mathcal{U}$ be a packing of type $(1+\varepsilon)\mathbb{D}^{-1}$ for some $\mathbb{D} \subset \mathbb{Z}_{\geq 2}$ and $\varepsilon > 0$. Let

$$K(\mathcal{U}) = \{k \in \mathbb{D} \colon \mathcal{U} \text{ contains a copy of } Q_k^d(\varepsilon)\} \tag{8}$$

and

$$k_{\max}(\mathcal{U}) = \max\{k \colon k \in K(\mathcal{U})\}. \tag{9}$$

For every $k \in K(\mathcal{U})$, let

$$\nu_k(\mathcal{U}) \text{ be the total number of copies of } Q_k^d(\varepsilon) \text{ in } \mathcal{U}. \tag{10}$$

Clearly, we have $0 \leq \nu_k(\mathcal{U}) \leq (k-1)^d$ for every k (recall that we suppose $\varepsilon > 0$). Finally, we define the *weight* of $\mathcal{U}$ as

$$\mathrm{w}(\mathcal{U}) = \sum_{k \in K(\mathcal{U})} (k-1)^{-d} \nu_k(\mathcal{U}). \tag{11}$$

We shall be interested in packings $\mathcal{U}$ with large weight. In that direction, we prove the following two technical results that are the core of our contribution. The first is essential to derive the lower bound for the online bounded space d-hypercube bin packing problem (Theorem 1) and a lower bound for PoA (Theorem 2); the second is essential to derive a lower bound for SPoA (Theorem 3). We remark that the technique of using weight functions in the analysis of packing algorithms dates back to the seventies (see [12] and the references therein).

Lemma 1 (Packing Lemma A). *There is an absolute constant d_0 for which the following holds for any $d \geq d_0$. Let*

$$S = \left\lceil \frac{2d}{9 \log d} \right\rceil . \tag{12}$$

The unit bin admits a packing $\mathcal{U}$ of type $(1 + S^{-2})\mathbb{Z}_{\geq 2}^{-1}$ with $k_{\max}(\mathcal{U}) = S$ and with

$$\mathrm{w}(\mathcal{U}) \geq \frac{d}{5 \log d}. \tag{13}$$

Lemma 2 (Packing Lemma B). *There is an absolute constant d_0 for which the following holds for any $d \geq d_0$. Let*

$$S' = \lceil \log_2 d - \log_2 \log d - 3 \rceil \tag{14}$$

and $\varepsilon = 2^{-2(S'-1)}$. The unit bin admits a packing $\mathcal{U}$ of type $(1 + \varepsilon)\mathbb{Z}_{2^+}^{-1}$ with $k_{\max}(\mathcal{U}) = 2^{S'-1}$ and with $\mathrm{w}(\mathcal{U}) \geq \log d$.

The two lemmas stated in this section are central to the proofs of Theorems 1, 2 and 3, which depend on further auxiliary results as shown in Fig. 1.

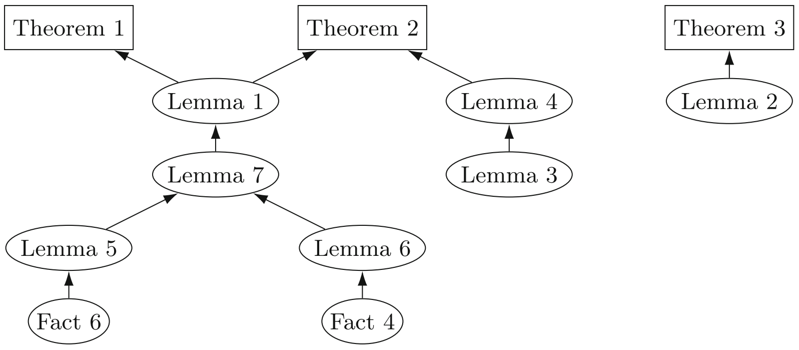


Fig. 1. A diagram illustrating the flow of the proofs of our main results

3 Proofs of Theorems 1 and 2

Proof of Theorem 1. Let $\mathcal{A}$ be any algorithm for the online bounded space d-hypercube bin packing problem. Let M be the maximum number of bins that $\mathcal{A}$ is allowed to leave open during its execution. To prove that $\mathcal{A}$ has asymptotic performance ratio $\Omega(d/\log d)$, we construct a suitable instance $\mathcal{I}$ for $\mathcal{A}$.

Let a packing $\mathcal{U}$ as in the statement of Lemma 1 be fixed. The instance $\mathcal{I}$ will be constructed by choosing a suitable integer N and then arranging the hypercubes in $2MN$ copies of $\mathcal{U}$ in a linear order, with all the hypercubes of the same size appearing together. Let us now formally describe $\mathcal{I}$.

Let

$$N = \prod_{k \in K(\mathcal{U})} (k-1)^d \tag{15}$$

and, for every $k \in K(\mathcal{U})$, let

$$N_{\widehat{k}} = \frac{N}{(k-1)^d} = \prod_{k' \in K(\mathcal{U}) \smallsetminus \{k\}} (k'-1)^d. \tag{16}$$

Recall that $\mathcal{U}$ contains $\nu_k(\mathcal{U})$ copies of $Q_k^d(\varepsilon)$ for every $k \in K(\mathcal{U})$. Let $K = |K(\mathcal{U})|$ and suppose $K(\mathcal{U}) = \{k_1, \dots, k_K\}$. The instance $\mathcal{I}$ that we shall construct is the concatenation of K segments, say $\mathcal{I} = \mathcal{I}_1 \dots \mathcal{I}_K$, with each segment $\mathcal{I}_\ell$ ($1 \le \ell \le K$) composed of a certain number of copies of $Q_{k_\ell}^d(\varepsilon)$. For every $1 \le \ell \le K$, set

$$f(\ell) = 2MN\nu_{k_\ell}(\mathcal{U}), \tag{17}$$

and

$$\mathcal{I}_\ell = (Q_{k_\ell}^d(\varepsilon), \dots, Q_{k_\ell}^d(\varepsilon)) = Q_{k_\ell}^d(\varepsilon)^{f(\ell)}. \tag{18}$$

That is, $\mathcal{I}_\ell$ is composed of a sequence of $f(\ell)$ copies of $Q_{k_\ell}^d(\varepsilon)$. This completes the definition of our instance $\mathcal{I}$.

Let us first state the following fact concerning the offline packing of the hypercubes in $\mathcal{I}$. This fact is clear, as we obtained $\mathcal{I}$ by rearranging the hypercubes in $2MN$ copies of $\mathcal{U}$.

Fact 4. *The hypercubes in $\mathcal{I}$ can be packed into at most $2MN$ unit bins.*

We now prove that, when $\mathcal{A}$ is given the instance $\mathcal{I}$ above, it will have performance ratio at least as bad as $\mathrm{w}(\mathcal{U})/2$. In view of (13) in Lemma 1, this will complete the proof of Theorem 1.

Let us examine the behaviour of $\mathcal{A}$ when given input $\mathcal{I}$. Fix $1 \le \ell \le K$ and suppose $\mathcal{A}$ has already seen the hypercubes in $\mathcal{I}_1 \dots \mathcal{I}_{\ell-1}$ and it has already packed them somehow. We now consider what happens when $\mathcal{A}$ examines the $f(\ell)$ hypercubes in $\mathcal{I}_\ell$, which are all copies of $Q_{k_\ell}^d(\varepsilon)$.

Clearly, since $\varepsilon > 0$, the $f(\ell)$ copies of $Q_{k_\ell}^d(\varepsilon)$ in $\mathcal{I}_\ell$ cannot be packed into fewer than

$$\frac{f(\ell)}{(k_\ell - 1)^d} = \frac{2MN\nu_{k_\ell}(\mathcal{U})}{(k_\ell - 1)^d} = 2MN_{\widehat{k}}\nu_{k_\ell}(\mathcal{U}) \ge MN_{\widehat{k}}\nu_{k_\ell}(\mathcal{U}) + M \tag{19}$$

unit bins. Therefore, even if some hypercubes in $\mathcal{I}_\ell$ are placed in bins still left open after the processing of $\mathcal{I}_1 \dots \mathcal{I}_{\ell-1}$, the hypercubes in $\mathcal{I}_\ell$ will add at least $MN_{\widehat{k}}\nu_{k_\ell}(\mathcal{U})$ new bins to the output of $\mathcal{A}$. Thus, the total number of bins that $\mathcal{A}$ will use when processing $\mathcal{I}$ is at least

$$\sum_{k \in K(\mathcal{U})} MN_{\widehat{k}}\nu_k(\mathcal{U}) = MN \sum_{k \in K(\mathcal{U})} (k-1)^{-d}\nu_k(\mathcal{U}) = MN\,\mathrm{w}(\mathcal{U}). \tag{20}$$

In view of Fact 4, it follows that the asymptotic performance ratio of $\mathcal{A}$ is at least

$$\frac{MN\,\mathrm{w}(\mathcal{U})}{2MN} = \frac{1}{2}\,\mathrm{w}(\mathcal{U}), \tag{21}$$

as required. This completes the proof of Theorem 1.

Proof of Theorem 2. Besides Lemma 1, we shall use the next two lemmas, the proofs of which are presented in [17].

Lemma 3. *Let $d \geq 2$ and $\varepsilon > 0$ be given. Any packing $\mathcal{P} = (\mathcal{U}_1, \mathcal{U}_2, \dots)$ of type $\mathcal{H}^d(\varepsilon)$ is a Nash equilibrium.*

Lemma 4. *If $\mathcal{U}$ is a packing of d-hypercubes into a unit bin of type $(1+\varepsilon)\mathbb{Z}_{\geq 2}^{-1}$, where*

$$0 < \varepsilon \leq \frac{1}{k_{\max}(\mathcal{U}) - 1}, \tag{22}$$

then $\mathrm{PoA}(d) \geq \mathrm{w}(\mathcal{U})$.

Let d_0 be as in Lemma 1 and suppose $d \geq d_0$. Moreover, let $\mathcal{U}$ be as given in that lemma. We now invoke Lemma 4 with $\varepsilon = S^{-2}$. Note that condition (22) does hold, as $\varepsilon = S^{-2} \leq 1/(S-1) = 1/(k_{\max}(\mathcal{U}) - 1)$. Combining Lemmas 1 and 4, we conclude that $\mathrm{PoA}(d) \geq \mathrm{w}(\mathcal{U}) \geq d/5 \log d$.

4 Proof of Lemma 1

We shall describe packings in terms of words of certain languages. For that, we define the languages we are interested in, show the properties we require, and then prove their existence. Owing to space limitation, we present only an outline of the proof of Lemma 1.

4.1 Separated Families of Languages

Let an integer $d \geq 2$ be fixed. We consider sets of words $L_k \subset [k]^d = \{1, \dots, k\}^d$ for $k \geq 2$. We refer to such L_k as *languages* or *k-languages*. Such languages L_k will specify 'positions' where we shall place $Q_k^d(\varepsilon)$ in certain packings (roughly speaking, for each $w \in L_k$, we put a certain copy $Q(w)$ of $Q_k^d(\varepsilon)$ in our packings (see (25)–(31) for the definition of $Q(w)$)).

We now introduce some conditions on the L_k that will help us make sure that we have a packing when we consider the $Q(w)$ $(w \in L_k)$ all together.

Definition 4 (Gapped languages). *Suppose $k \geq 2$ and let a k-language $L_k \subset [k]^d$ be given. We say that L_k misses j at coordinate i_0 if every word $w = (w_i)_{1 \leq i \leq d}$ in L_k is such that $w_{i_0} \neq j$. Furthermore, L_k is said to be gapped if, for each $1 \leq i \leq d$, either L_k misses $k-1$ at i or L_k misses k at i.*

The reason we are interested in gapped languages is as follows. Suppose L_k is a gapped language as in Definition 4, and suppose $w = (w_i)_{1\le i\le d}$ and $w' = (w'_i)_{1\le i\le d}$ are distinct words in L_k. Then $Q(w)$ and $Q(w')$ do not overlap (this can be checked from (30) and Fact 5(ii); see Lemma 6(i)). Thus, if we let $\mathcal{P}_k$ be the collection of the $Q(w)$ $(w \in L_k)$, then $\mathcal{P}_k$ is a packing. We now introduce a certain notion of 'compatibility' between two languages L_k and $L_{k'}$, so that $\mathcal{P}_k$ and $\mathcal{P}_{k'}$ can be put together to obtain a packing if they come from 'compatible' languages L_k and $L_{k'}$.

Definition 5 (Separated languages). *Suppose $2 \le k < k'$ and $L_k \subset [k]^d$ and $L_{k'} \subset [k']^d$ are given. We say that L_k and $L_{k'}$ are separated if, for any $w = (w_i)_{1\le i\le d} \in L_k$ and any $w' = (w'_i)_{1\le i\le d} \in L_{k'}$, there is some i such that $w_i < k < k' = w'_i$.*

Suppose L_k and $L_{k'}$ are gapped and separated. Consider the corresponding packings $\mathcal{P}_k$ and $\mathcal{P}_{k'}$ as above. Fact 5(i) and (30) imply that $\mathcal{P}_k \cup \mathcal{P}_{k'}$ is a packing. To check this, let $w = (w_i)_{1\le i\le d} \in L_k$ and any $w' = (w'_i)_{1\le i\le d} \in L_{k'}$ be given. Then, by definition, there is some i such that $w_i < k < k' = w'_i$. This implies that $Q(w) = Q^{(k)}(w)$ and $Q(w') = Q^{(k')}(w')$ are disjoint 'in the ith dimension' (see Fact 5(i) and Lemma 6(i)).

Definition 6 (Separated families). *Let $\mathcal{L} = (L_k)_{2\le k\le S}$ be a family of k-languages $L_k \subset [k]^d$. If, for every $2 \le k < k' \le S$, the languages L_k and $L_{k'}$ are separated, then we say that $\mathcal{L}$ is a separated family of languages.*

Remark 1. For $2 \le k \le d$, let $L_k = \{w = (w_i)_{1\le i\le k} \in [k]^d\colon w_k = k \text{ and } w_i < k \text{ for all } i \ne k\}$. One can then check that $\mathcal{L} = (L_k)_{2\le k\le d}$ is a family of gapped, separated languages. Consider the packing $\mathcal{P} = \bigcup_{2\le k\le d} \mathcal{P}_k$ with the $\mathcal{P}_k$ defined by the L_k as above. We have $\nu_k(\mathcal{P}) = |L_k| = (k-1)^{d-1}$ (recall (10)) and $\mathrm{w}(\mathcal{P}) = \sum_{2\le k\le d} 1/(k-1) \sim \log d$ (recall (11)). The existence of $\mathcal{P}$ implies a weak form of Theorem 1 (namely, a lower bound of $\Omega(\log d)$ instead of $\Omega(d/\log d)$).

Remark 1 above illustrates the use we wish to make of families of gapped, separated languages. Our focus will soon shift onto producing much 'better' families than the one explicitly defined in Remark 1. Indeed, the main result in this section is the following lemma, for which we give a probabilistic proof (see Sect. 5 and [17]).

Lemma 5 (Many large, separated gapped languages). *There is an absolute constant d_0 such that, for any $d \ge d_0$, there is a separated family $\mathcal{L} = (L_k)_{2\le k\le S}$ of gapped k-languages $L_k \subset [k]^d$ such that*

$$|L_k| \ge \frac{10}{11}(k-1)^d, \tag{23}$$

for every $2 \le k \le S$, where

$$S = \left\lceil \frac{2d}{9\log d} \right\rceil. \tag{24}$$

Fix $\mathcal{L} = (L_k)_{2\le k\le S}$ a family of separated, gapped k-languages $L_k \subset [k]^d$. We shall now give, for every sufficiently small $\varepsilon > 0$, the construction of a packing $\mathcal{U}_\varepsilon = \mathcal{U}_\varepsilon(\mathcal{L})$ of d-hypercubes into the unit bin $[0,1]^d$ using $\mathcal{L}$. Choosing $\mathcal{L}$ suitably, we shall be able to prove Lemma 7 below, which takes us very close to the proof of Lemma 1.

The packing $\mathcal{U}_\varepsilon$. The packing $\mathcal{U}_\varepsilon = \mathcal{U}_\varepsilon(\mathcal{L})$ contains copies of the hypercubes $Q_k^d(\varepsilon)$ for $2 \le k \le S$. In fact, for each $w \in L_k$ $(2 \le k \le S)$, we place a copy $Q(w)$ of $Q_k^d(\varepsilon)$ in $\mathcal{U}_\varepsilon$. To specify the location of the copy $Q(w)$ of $Q_k^d(\varepsilon)$ in $\mathcal{U}_\varepsilon$, we need a definition.

Definition 7 (Base point coordinates of the $Q(w)$). *For every $k \ge 2$ and $0 < \varepsilon < 1/(k-1)$, let*

$$x^{(k)}(j) = x_\varepsilon^{(k)}(j) = \begin{cases} \dfrac{j-1}{k_-} = \dfrac{(j-1)(1+\varepsilon)}{k}, & \text{if } 1 \le j < k \\ 1 - \dfrac{1}{k_-} = 1 - \dfrac{1+\varepsilon}{k}, & \text{if } j = k. \end{cases} \tag{25}$$

Moreover, for $1 \le j \le k$, let

$$y^{(k)}(j) = x^{(k)}(j) + \frac{1}{k_-} = x^{(k)}(j) + \frac{1+\varepsilon}{k}. \tag{26}$$

Note that, for each $2 \le k \le S$, we have

$$\begin{aligned} 0 = x^{(k)}(1) < y^{(k)}(1) = x^{(k)}(2) < y^{(k)}(2) = x^{(k)}(3) < \cdots < y^{(k)}(k-2) \\ = x^{(k)}(k-1) < x^{(k)}(k) < y^{(k)}(k-1) < y^{(k)}(k) = 1. \end{aligned} \tag{27}$$

For convenience, for every $k \ge 2$ and every $1 \le j \le k$, let

$$I^{(k)}(j) = (x^{(k)}(j), y^{(k)}(j)) \subset [0,1]. \tag{28}$$

Now, for each word $w = (w_i)_{1\le i\le d} \in L_k$ $(2 \le k \le S)$, let

$$x[w] = x^{(k)}[w] = (x^{(k)}(w_1), \dots, x^{(k)}(w_d)) \in \mathbb{R}^d, \text{and} \tag{29}$$

$$Q(w) = Q^{(k)}(w) = x^{(k)}[w] + Q_k^d(\varepsilon) \subset [0,1]^d. \tag{30}$$

Putting together the definitions, one checks that

$$\begin{aligned} Q(w) = Q^{(k)}(w) &= I^{(k)}(w_1) \times \cdots \times I^{(k)}(w_d) \\ &= \big(x^{(k)}(w_1), y^{(k)}(w_1)\big) \times \cdots \times \big(x^{(k)}(w_d), y^{(k)}(w_d)\big) \subset [0,1]^d. \end{aligned} \tag{31}$$

Definition 8 (Packing $\mathcal{U}_\varepsilon = \mathcal{U}_\varepsilon(\mathcal{L})$). *Suppose $\mathcal{L} = (L_k)_{2\le k\le S}$ is a family of separated, gapped k-languages $L_k \subset [k]^d$. Let $0 < \varepsilon \le S^{-2}$. Define the packing $\mathcal{U}_\varepsilon = \mathcal{U}_\varepsilon(\mathcal{L})$ as follows. For each $2 \le k \le S$ and each $w \in L_k$, place the copy $Q(w) = Q^{(k)}(w) \subset [0,1]^d$ of $Q_k^d(\varepsilon)$ in $\mathcal{U}_\varepsilon$.*

To prove that $\mathcal{U}_\varepsilon$ is indeed a packing, that is, that the hypercubes in $\mathcal{U}_\varepsilon$ are pairwise disjoint, we use the following fact (see [17]).

Fact 5. *The following assertions hold.*

(i) Suppose $2 \le k < k' \le S$ and $0 < \varepsilon \le S^{-2}$. Then

$$y^{(k)}(k-1) < x^{(k')}(k'). \tag{32}$$

In particular, the intervals $I^{(k)}(j)$ $(1 \le j < k)$ are disjoint from $I^{(k')}(k')$.

(ii) For any $2 \le k \le S$, the intervals $I^{(k)}(j)$ $(1 \le j \le k)$ are pairwise disjoint, except for the single pair formed by $I^{(k)}(k-1)$ and $I^{(k)}(k)$.

For the next lemma, recall (8) and (10), and Definition 3.

Lemma 6. *Suppose $\mathcal{L} = (L_k)_{2 \le k \le S}$ is a family of separated, non-empty gapped k-languages $L_k \subset [k]^d$. Suppose $0 < \varepsilon \le S^{-2}$. Let $\mathcal{U}_\varepsilon = \mathcal{U}_\varepsilon(\mathcal{L})$ be the family of all the hypercubes $Q(w) = Q^{(k)}(w) \subset [0,1]^d$ with $w \in L_k$ and $2 \le k \le S$. Then the following assertions hold:* (i) *the hypercubes in $\mathcal{U}_\varepsilon$ are pairwise disjoint and form a packing of type $(1+\varepsilon)\mathbb{Z}_{\ge 2}^{-1}$;* (ii) *for every $2 \le k \le S$, we have $\nu_k(\mathcal{U}_\varepsilon) = |L_k|$;* (iii) *$|K(\mathcal{U}_\varepsilon)| = S - 1$.*

Lemma 7. *There is an absolute constant d_0 for which the following holds for any $d \ge d_0$. Let $S = \lceil 2d/(9 \log d) \rceil$. The unit bin admits a packing $\mathcal{U}$ of type $(1 + S^{-2})\mathbb{Z}_{\ge 2}^{-1}$ and with $k_{\max}(\mathcal{U}) = S$ such that $\mathrm{w}(\mathcal{U}) \ge (10/11)(S-1)$.*

Lemma 7 is an immediate corollary of Lemmas 5 and 6. From it, the proof of Lemma 1 follows easily: taking the packing $\mathcal{U}$ given in this lemma, we have that $\mathrm{w}(\mathcal{U}) \ge (10/11)(S-1) \ge d/(5 \log d)$, as long as d is large enough.

5 Proof of Lemma 5

We need the following auxiliary fact, which follows from standard Chernoff bounds for the hypergeometric distribution.

Fact 6. *There is an absolute constant d_0 such that, for any $d \ge d_0$, there are sets $F_1, \dots, F_d \subset [d]$ such that* (i) *for every $1 \le k \le d$, we have $|F_k| = \lceil d/2 \rceil$ and* (ii) *for every $1 \le k < k' \le d$, we have $|F_k \cap F_{k'}| < 7d/26$.*

We now proceed to prove Lemma 5. Let $S = \lceil 2d/9 \log d \rceil$ and let $F_1, \dots, F_d$ be as in Fact 6. In what follows, we only use F_k for $2 \le k \le S$. For each $2 \le k \le S$, we shall construct $L_k \subset [k]^d$ in two parts. First, let

$$L'_k \subset ([k] \smallsetminus \{k-1\})^{F_k} = \{w = (w_i)_{i \in F_k} : w_i \in [k] \smallsetminus \{k-1\} \text{ for all } i \in F_k\} \tag{33}$$

and then set

$$\begin{aligned} L_k &= L'_k \times [k-1]^{[d] \smallsetminus F_k} \\ &= \{w = (w_i)_{1 \le i \le d} \colon \exists w' = (w'_i)_{i \in F_k} \in L'_k \text{ such that } w_i = w'_i \text{ for all } i \in F_k \\ &\qquad \text{and } w_i \in [k-1] \text{ for all } i \in [d] \smallsetminus F_k\}. \end{aligned} \tag{34}$$

Note that, by (33) and (34), the k-language L_k will be gapped ($k-1$ is missed at every $i \in F_k$ and k is missed at every $i \in [d] \setminus F_k$). We shall prove that there is a suitable choice for the L'_k with $|L'_k| \geq (10/11)(k-1)^d$ ensuring that $\mathcal{L} = (L_k)_{2 \leq k \leq S}$ is separated. Since we shall then have

$$|L_k| = |L'_k|(k-1)^{d-|F_k|} \geq \frac{10}{11}(k-1)^d, \tag{35}$$

condition (23) will be satisfied. We now proceed with the construction of the L'_k.

Fix $2 \leq k \leq S$. For $2 \leq \ell < k$, let

$$J(\ell, k) = F_k \setminus F_\ell, \tag{36}$$

and note that

$$|J(\ell, k)| > \left\lceil \frac{d}{2} \right\rceil - \frac{7}{26}d \geq \frac{3}{13}d. \tag{37}$$

Let $v = (v_i)_{i \in F_k}$ be an element of $([k] \setminus \{k-1\})^{F_k}$ chosen uniformly at random. For every $2 \leq \ell < k$, we say that v is *ℓ-bad* if $v_i \neq k$ for every $i \in J(\ell, k)$. Moreover, we say that v is *bad* if it is ℓ-bad for some $2 \leq \ell < k$. It is clear that

$$\mathbb{P}(v \text{ is } \ell\text{-bad}) = \left(1 - \frac{1}{k-1}\right)^{|J(\ell,k)|} \leq \mathrm{e}^{-|J(\ell,k)|/S} \leq \exp\left(-\frac{3d}{13\lceil 2d/9 \log d\rceil}\right) \leq d^{-1}, \tag{38}$$

for every large enough d, whence

$$\mathbb{P}(v \text{ is bad}) \leq Sd^{-1} \leq \frac{1}{4 \log d} \leq \frac{1}{11} \tag{39}$$

if d is large enough. Therefore, at least $(10/11)(k-1)^{|F_k|}$ words $v \in ([k] \setminus \{k-1\})^{F_k}$ are not bad, as long as d is large enough. We let $L'_k \subset ([k] \setminus \{k-1\})^{F_k}$ be the set of such good words. The following claim completes the proof of Lemma 5.

Claim. With the above choice of L'_k ($2 \leq k \leq S$), the family $\mathcal{L} = (L_k)_{2 \leq k \leq S}$ of the languages L_k as defined in (34) is separated.

Proof. Fix $2 \leq \ell < k \leq S$. We show that L_ℓ and L_k are separated. Let $u = (u_i)_{1 \leq i \leq d} \in L_\ell$ and $w = (w_i)_{1 \leq i \leq d} \in L_k$ be given. By the definition of L_k, there is $v = (v_i)_{i \in F_k} \in L'_k$ such that $w_i = v_i$ for all $i \in F_k$. Furthermore, since $v \in L'_k$ is not a bad word, it is not ℓ-bad. Therefore, there is $i_0 \in J(\ell, k) = F_k \setminus F_\ell$ for which we have $v_{i_0} = k$. Observing that $i_0 \notin F_\ell$ and recalling the definition of L_ℓ, we see that $u_{i_0} < \ell < k = v_{i_0} = w_{i_0}$, as required.

Acknowledgements. We thank the referees for helpful suggestions.

References

1. Aumann, R.J.: Acceptable points in general cooperative n-person games. In: Luce, R.D., Tucker, A.W. (eds.) Contribution to the Theory of Game IV, Annals of Mathematical Study, vol. 40, pp. 287–324. University Press (1959)
2. Balogh, J., Békési, J., Dósa, G., Epstein, L., Levin, A.: Lower bounds for several online variants of bin packing. http://arxiv.org/abs/1708.03228 (2017)
3. Balogh, J., Békési, J., Galambos, G.: New lower bounds for certain classes of bin packing algorithms. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 25–36. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18318-8_3
4. Bilò, V.: On the packing of selfish items. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium, pp. 9–18. IEEE (2006)
5. Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: a survey, chap. 2. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS (1997)
6. Csirik, J.: An on-line algorithm for variable-sized bin packing. Acta Inform. **26**(8), 697–709 (1989)
7. Csirik, J., van Vliet, A.: An on-line algorithm for multidimensional bin packing. Oper. Res. Lett. **13**, 149–158 (1993)
8. Epstein, L.: Bin packing games with selfish items. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 8–21. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40313-2_2
9. Epstein, L., Kleiman, E.: Selfish bin packing. Algorithmica **60**(2), 368–394 (2011)
10. Epstein, L., van Stee, R.: Bounds for online bounded space hypercube packing. Discrete Optim. **4**(2), 185–197 (2007)
11. Epstein, L., Kleiman, E., Mestre, J.: Parametric packing of selfish items and the subset sum algorithm. Algorithmica **74**(1), 177–207 (2016)
12. Epstein, L., van Stee, R.: Optimal online algorithms for multidimensional packing problems. SIAM J. Comput. **35**(2), 431–448 (2005)
13. Fernandes, C.G., Ferreira, C.E., Miyazawa, F.K., Wakabayashi, Y.: Selfish square packing. In: Proceedings of the VI Latin-American Algorithms, Graphs and Optimization Symposium. Electronic Notes in Discrete Mathematics, vol. 37, pp. 369–374 (2011)
14. Fernandes, C.G., Ferreira, C.E., Miyazawa, F.K., Wakabayashi, Y.: Prices of anarchy of selfish 2D bin packing games. http://arxiv.org/abs/1707.07882 (2017)
15. Heydrich, S., van Stee, R.: Beating the harmonic lower bound for online bin packing. In: ICALP 2016. LIPIcs, vol. 55, pp. 41:1–41:14. Dagstuhl, Germany (2016). http://arxiv.org/abs/1511.00876v5
16. Heydrich, S., van Stee, R.: Improved lower bounds for online hypercube packing. http://arxiv.org/abs/1607.01229 (2016)
17. Kohayakawa, Y., Miyazawa, F.K., Wakabayashi, Y.: A tight lower bound for an online hypercube packing problem and bounds for prices of anarchy of a related game. http://arxiv.org/abs/1712.06763 (2017)
18. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49116-3_38
19. Ma, R., Dósa, G., Han, X., Ting, H.F., Ye, D., Zhang, Y.: A note on a selfish bin packing problem. J. Glob. Optim. **56**(4), 1457–1462 (2013)
20. Nash, J.: Non-cooperative games. Ann. Math. **54**(2), 286–295 (1951)

21. Seiden, S.S.: An optimal online algorithm for bounded space variable-sized bin packing. SIAM J. Discrete Math. **14**, 458–470 (2001)
22. Seiden, S.S.: On the online bin packing problem. J. Assoc. Comput. Mach. **49**(5), 640–671 (2002)
23. van Vliet, A.: An improved lower bound for online bin packing algorithms. Inf. Process. Lett. **43**, 277–284 (1992)
24. Yu, G., Zhang, G.: Bin packing of selfish items. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 446–453. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92185-1_50

The Parameterized Complexity of Cycle Packing: Indifference is Not an Issue

R. Krithika[1,2], Abhishek Sahu[1,2(✉)], Saket Saurabh[1,2,3], and Meirav Zehavi[4]

[1] The Institute of Mathematical Sciences, HBNI, Chennai, India
{rkrithika,asahu,saket}@imsc.res.in
[2] UMI ReLax, Chennai, India
[3] University of Bergen, Bergen, Norway
[4] Ben-Gurion University, Beersheba, Israel
meiravze@bgu.ac.il

Abstract. In the Cycle Packing problem, we are given an undirected graph G, a positive integer r, and the task is to check whether there exist r vertex-disjoint cycles. In this paper, we study Cycle Packing with respect to a structural parameter, namely, distance to proper interval graphs (indifference graphs). In particular, we show that Cycle Packing is fixed-parameter tractable (FPT) when parameterized by t, the size of a proper interval deletion set. For this purpose, we design an algorithm with $\mathcal{O}(2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)})$ running time. Several structural parameterizations for Cycle Packing have been studied in the literature and our FPT algorithm fills a gap in the ecology of such parameterizations. We combine color coding, greedy strategy and dynamic programming based on structural properties of proper interval graphs in a non-trivial fashion to obtain the FPT algorithm.

1 Introduction

Packing problems form a fundamental class of optimization problems in computer science. They involve finding a collection of objects with certain properties – examples include Bin Packing, Knapsack, Independent Set and Cycle Packing. Here, we focus on the Cycle Packing problem in the realm of parameterized complexity. In the Cycle Packing problem, we are given an undirected graph G and a positive integer r, and the task is to check whether there exist r vertex-disjoint cycles. Since the publication of the classic Erdös-Pósa theorem in 1965 [13], this problem has received significant scientific attention in the fields of Graph Theory and Algorithm Design. In particular, Cycle Packing is one of the first problems studied in the framework of parameterized complexity. In this framework, each problem instance is associated with a non-negative integer k called *parameter*, and a problem is said to be *fixed-parameter tractable* (FPT) if it can be solved in $f(k)n^{\mathcal{O}(1)}$ time for some function f, where n is the input size.

Due to space limitations, most proofs have been omitted.

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 712–726, 2018.
https://doi.org/10.1007/978-3-319-77404-6_52

For convenience, the running time $f(k)n^{\mathcal{O}(1)}$ where f grows superpolynomially with k is denoted as $\mathcal{O}^*(f(k))$. Further details on parameterized algorithms can be found in [9,12,16].

In the standard parameterization of Cycle Packing, the parameter is the number r of vertex-disjoint cycles. The non-uniform fixed-parameter tractability of Cycle Packing follows from the Robertson-Seymour theorem [33], a fact observed by Fellows and Langston in the 1980s [14]. In 1994, Bodlaender showed that Cycle Packing can be solved in $\mathcal{O}^*(2^{\mathcal{O}(r^2)})$ time [2]. A *feedback vertex set* is a set of vertices whose deletion results in a forest. The Erdös-Pósa theorem states that there exists a function $f(r) = \mathcal{O}(r \log r)$ such that for each non-negative integer r, every undirected graph either contains r vertex-disjoint cycles or has a feedback vertex set consisting of $f(r)$ vertices [13]. It is well known that the treewidth (tw) of a graph is not larger than the size of its feedback vertex set, and that a naive dynamic programming scheme solves Cycle Packing in $\mathcal{O}^*(2^{\mathcal{O}(tw \log tw)})$ time (see, e.g., [9]). Thus, the existence of an $\mathcal{O}^*(2^{\mathcal{O}(r \log^2 r)})$ time algorithm can be viewed as a direct consequence of the Erdös-Pósa theorem. Recently, Lokshtanov et al. [27] obtained an algorithm with running time $\mathcal{O}^*(2^{\mathcal{O}(\frac{r \log^2 r}{\log \log r})})$ for Cycle Packing, improving upon the classical consequence of the Erdös-Pósa theorem. However, the focus of this paper is to study Cycle Packing with respect to *structural parameters* rather than *solution size*.

Structural Parameterizations. In the early years of parameterized complexity and algorithms, problems were almost always parameterized by the solution size. Recent research has focused on other parameterizations based on structural parameters in the input [23], or above or below some guaranteed optimum values [20,21,28]. Such 'non-standard' parameters are more likely to be small in practice. Also, once a problem is shown to be FPT or to have a polynomial sized kernel by a parameterization, it is natural to ask whether the problem is FPT (and admits a polynomial kernel) when parameterized by a provably smaller parameter. In the same vein, if we show that a problem is W-hard under a parameterization, it is natural to ask whether it is FPT when parameterized by a provably larger parameter.

Apart from solution size, *treewidth* is one of the most well studied parameters. However, in the context of Cycle Packing, our understanding of treewidth is *complete* in the following sense: While Cycle Packing is known to be solvable in time $\mathcal{O}^*(2^{\mathcal{O}(tw \log tw)})$, it cannot be solved in time $\mathcal{O}^*(2^{o(tw \log tw)})$ unless the Exponential Time Hypothesis fails [10]. Another parameter that has gained significant attention recently is the size of a *modulator* to a family of graphs. Let $\mathcal{F}$ be a family of graphs. Given a graph G and a set $S \subseteq V(G)$, we say that S is an *$\mathcal{F}$-modulator* if $G - S$ is in $\mathcal{F}$. For example, if $\mathcal{F}$ is the family of independent sets, forests, bipartite graphs, interval graphs and chordal graphs, then the modulator corresponds to a vertex cover, feedback vertex set, odd cycle transversal, interval deletion set and chordal deletion set, respectively. The size of S is also called the *vertex-deletion distance* to $\mathcal{F}$. One of the earliest studies in the realm of alternate parameterizations is by Cai [6]. Cai [6] studied Coloring problems parameterized by the vertex-deletion distance to various graph classes including bipartite graphs and split graphs. Fellows et al. [15] studied alternate parameterizations for problems that were proven to be intractable with respect to the standard parameterization. This led to a whole new ecology program and opened up a floodgate of new and exciting research. Structural parameterizations of the classical Vertex Cover ([3,23]) and Feedback Vertex Set [24] have also been explored. We refer to [23] for a detailed introduction to the whole program as well as the thesis of Jansen [22].

Focusing on structural parameters for Cycle Packing, the main topic of this paper, Bodlaender et al. [4] obtained polynomial kernels with respect to the size of a vertex cover, the vertex-deletion distance to a cluster graph and the maximum leaf number (see [4] or [22] for definitions). There is also a kernel lower bound result known for parameterization with respect to solution size [5] leading to several other lower bounds.

Our Choice of Parameter and Our Result. In order for an FPT algorithm to exist for a parameterized problem, it is necessary that it must be solvable in polynomial time when the parameter is a constant (in particular, zero). Since Cycle Packing is solvable in $\mathcal{O}^*(2^{\mathcal{O}(tw \log tw)})$ time on graphs of treewidth tw, we have that Cycle Packing is FPT parameterized by vertex cover size, feedback vertex set size, pathwidth, and vertex-deletion distance to graphs of constant treewidth. However, the status of the problem when parameterized by the vertex-deletion distance to interval graphs or chordal graphs has not yet been studied. Cycle Packing is NP-complete on chordal graphs [19] and thus we cannot hope to have an algorithm with running time $n^{f(t)}$, where t is the size of the modulator to chordal graphs, unless P = NP. On the other hand, the classical complexity status of Cycle Packing on interval graphs is not known. That is, we do not know whether Cycle Packing admits a polynomial time algorithm on interval graphs.[1] A natural graph class that is a subset of the class of interval graphs is the one of *proper interval graphs* (also known as *indifference graphs* and *unit interval graphs* in the literature). A graph is a proper interval graph if its vertices can be assigned to intervals such that there is an edge between two vertices if and only if their corresponding intervals have non-empty intersection. Further, this set of intervals should satisfy the property that no interval properly contains another. It is well known that Cycle Packing can be solved in polynomial time on proper interval graphs [31]. This is the starting point of our work.

We consider Cycle Packing parameterized by the size of an $\mathcal{F}$-modulator where $\mathcal{F}$ is the set of all proper interval graphs. Cycle Packing is simply Triangle Packing in proper interval graphs (and more generally in chordal graphs). Interval graphs and proper interval graphs have a rich geometric structure which makes them amenable to efficient algorithms for most classical problems [18]. They have applications in several fields like scheduling, archaeology,

[1] The algorithm described in [26] does not seem to be correct as it does not produce the correct answer for a family of input instances.

developmental psychology and DNA sequencing [18]. Also, interval graphs and proper interval graphs are well studied in the framework of parameterized algorithms in the context of vertex-deletion problems [7,8,17,25,34,35]. A set of vertices is called a *proper interval deletion set* if its deletion results in a proper interval graph. By parameterizing Cycle Packing with respect to the size of such a set as parameter, we attempt to understand the complexity of the problem on *almost proper interval graphs*. The main result of the paper is the following theorem.

Theorem 1. Cycle Packing *parameterized by the size t of a proper interval deletion set can be solved in $\mathcal{O}^*(2^{\mathcal{O}(t \log t)})$ time.*

We assume that the proper interval deletion set T is part of the input. This assumption is reasonable as given a graph G and an integer t, there is an algorithm that, in $\mathcal{O}^*(6^t)$ time, outputs a proper interval deletion set of size at most t (if one exists) [7,35].

Overview of Our Technique. Our FPT algorithm combines various ingredients like color coding, greedy strategy and a multi-layered dynamic programming routine. It crucially uses the properties of a *proper interval ordering* and a *clique partition* of a proper interval graph. Essentially, we reduce the problem of finding cycles in G to finding appropriate paths in the proper interval graph $H = G - T$. Our approach consists of (i) a guessing phase, where we determine important relations between the T and H. This allows us to replace the vertices in T by variables that capture precisely the roles of those vertices; (ii) a coloring phase, which allows us to separate the tasks associated with individual variables, that later enables us to employ a greedy strategy; (iii) a dynamic programming routine over the clique partition of H that incorporates a greedy strategy to find an assignment to these variables. In the first phase, we reduce Cycle Packing to multiple instances of a constraint satisfaction problem which we call Constrained Path Assignment using the existence of a solution with *nice properties*. In the second phase, we reduce Constrained Path Assignment to a 'colored' variant called Colorful Constrained Path Assignment using color coding. By looking for only colorful solutions, we not only reduce the search space of solutions but also make the assignment to two variables independent of each other. This independence allows us to use a greedy strategy in the next phase where we only look for a *canonical solution*. Informally, a canonical solution is a colorful solution that is *aligned* to the left or to the right with respect to the proper interval ordering of H. The third and final phase is an algorithm to find a canonical solution, if it exists, thereby solving Colorful Constrained Path Assignment. This is indubitably the most technical part of the FPT algorithm that employs a dynamic programming routine incorporating a greedy strategy over the clique partition of H. The greedy strategy is of the flavour "choose the leftmost/rightmost vertex from the vertices belonging to some specific restriction of a color set". This choice follows from the definition of a canonical solution.

Kernelization Complexity. The kernel lower bound result known for Cycle Packing with respect to the solution size r is by a reduction from Disjoint Factors [5]. This reduction produces instances where the underlying graph has a proper interval vertex deletion set of size r. Therefore, it follows that Cycle Packing parameterized by the size of a proper interval vertex deletion set does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Preliminaries. For graph theoretic terms not defined here, refer to [11,18]. The set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. The length of a path is defined as the number of vertices in it. For a collection of paths $\mathcal{P}$, $V(\mathcal{P})$ denotes $\bigcup_{P \in \mathcal{P}} V(P)$. The vertices of a proper interval graph H can be ordered by a permutation $\pi : V(H) \rightarrow [|V(H)|]$, called *proper interval ordering*, with the following property.

Proposition 1 ([30]). *Let H be a proper interval graph with proper interval ordering π. For every pair u, v of vertices with $\pi(u) < \pi(v)$, if $uv \in E(H)$, then $\{w \in V(H) \mid \pi(u) \leq \pi(w) \leq \pi(v)\}$ is a clique in H.*

Proposition 2 ([25]). *Given a proper interval graph H with proper interval ordering π, there is a linear-time algorithm that outputs a partition of $V(H)$ into an ordered set $\{Q_1, \ldots, Q_q\}$ of (pairwise disjoint) cliques (called* clique partition*) such that for each pair of vertices $u \in Q_i$, $v \in Q_j$ with $1 \leq i < j \leq q$, $\pi(v) > \pi(u)$. Further, for all $uv \in E(H)$, there is an index $i \in [q]$ such that either $u, v \in Q_i$ or $u \in Q_i$ and $v \in Q_{i+1}$.*

2 Phases 1 and 2: Reducing to Colorful Constrained Path Assignment

Let $\mathcal{I} = (G, T, r)$ denote the input instance of Cycle Packing. First, we define the notion of a *nice set of cycles.*

Definition 1 (Nice set of cycles). *Let $T \subseteq V(G)$ be a proper interval deletion set of a graph G. Let $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ be a clique partition of $H = G - T$. A set $\mathcal{C} = \{C_1, \ldots, C_r\}$ of vertex-disjoint cycles in G is a* nice set of cycles *if,*

- *For each $i \in [r]$, if $V(C_i) \subseteq V(H)$, then C_i is a triangle.*
- *For each $i \in [r]$ with $V(C_i) \cap T \neq \emptyset$, if C_i has a path P with $V(P) \subseteq V(H)$, then for each $j \in [q]$, $|V(P) \cap Q_j| \leq 2$.*
- *For each $j \in [q]$, the number of maximal paths P of length at least 3 that are contained in some cycle in $\mathcal{C}$ such that $V(P) \subseteq V(H)$ and $V(P) \cap Q_j \neq \emptyset$ is at most 14.*

Lemma 1. *Let $T \subseteq V(G)$ be a proper interval deletion set of a graph G. Let H be the proper interval graph $G - T$ with clique partition $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$. Given a set $\mathcal{C}$ of r vertex-disjoint cycles, a nice set $\mathcal{C}^*$ of r vertex-disjoint cycles can be obtained in polynomial time.*

Proof. The first two properties are easy to achieve by a simple exchange argument as every induced cycle in H is a triangle and every $Q \in \mathcal{Q}$ is a clique. For the third property, consider a clique Q_j where $j \in [q]$. Let $\mathcal{P}$ be the set of maximal paths of length at least 3 that are contained in some cycle in $\mathcal{C}$ such that for each $P \in \mathcal{P}$, $V(P) \subseteq V(H)$ and $V(P) \cap Q_j \neq \emptyset$. From Proposition 2, each path in $\mathcal{P}$ is in at least one of the following subsets.

- $\mathcal{P}_1 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 1, |V(P) \cap Q_{j-1}| \geq 1, |V(P) \cap Q_{j+1}| \geq 1\}$.
- $\mathcal{P}_2 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 1, |V(P) \cap Q_{j-1}| \geq 1, |V(P) \cap Q_{j-2}| \geq 1\}$.
- $\mathcal{P}_3 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 1, |V(P) \cap Q_{j+1}| \geq 1, |V(P) \cap Q_{j+2}| \geq 1\}$.
- $\mathcal{P}_4 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 2, |V(P) \cap Q_{j-1}| \geq 1\}$.
- $\mathcal{P}_5 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 2, |V(P) \cap Q_{j+1}| \geq 1\}$.
- $\mathcal{P}_6 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 1, |V(P) \cap Q_{j-1}| = 2\}$.
- $\mathcal{P}_7 = \{P \in \mathcal{P} : |V(P) \cap Q_j| = 1, |V(P) \cap Q_{j+1}| = 2\}$.

By the pigeonhole principle, if $|\mathcal{P}| > 14$, then there is an index $i \in [7]$ such that $|\mathcal{P}_i| \geq 3$. Let P_1, P_2 and P_3 be three distinct paths in $\mathcal{P}_i$. Let C_1, C_2 and C_3 be the cycles (not necessarily distinct) in $\mathcal{C}$ that contain P_1, P_2 and P_3, respectively. Then, C_1, C_2 and C_3 can be replaced by three triangles in $\mathcal{C}$. This procedure can be applied for each $j \in [q]$. When this replacement can no longer be made, we have a set $\mathcal{C}^*$ of r vertex-disjoint cycles that satisfies the third property. Further, if $\mathcal{C}$ satisfies the first two properties, then $\mathcal{C}^*$ also satisfies them. Repeatedly applying this procedure for each $j \in [q]$, we obtain a set $\mathcal{C}^*$ of at least r vertex-disjoint cycles that is a nice set of cycles. □

Phase 1. Next, we guess how T interacts with a solution that is a nice set of cycles. Any solution consists of cycles of two types; those cycles that are entirely contained in H and those cycles that have a vertex from T. The number of cycles that contain a vertex from T is at most $|T|$. We first guess this number ℓ. Then, for each of the ℓ cycles C_i, we guess the vertices T_i from T that it contains and also the order in which they appear. This information is captured as a partition of T into ℓ ordered sets, T_1 to T_ℓ. Any cycle C_i that has $T_i \subseteq T$ contains a path between every pair of consecutive vertices of T_i with internal vertices from H. The total number of such paths in these ℓ cycles is $\mathcal{O}(|T|)$. We guess the number of internal vertices of each such path as being 0,1, 2 or at least 3. Paths of length at least 3 can be assumed to satisfy a certain condition (given by Definition 1) regarding their intersections, which we explicitly encode as a constraint. In practice, we only demand such paths to be of length at least 2, but still retain the intersections-related constraint. As these paths are all completely contained in H, we can delete T from G once the constraints to be satisfied are encoded.

For this encoding, we introduce 4 sets W, X, Y and Z of variables corresponding to placeholders for paths that we wish to find in H. For each path of length 1, there is a variable in Z, and for each path of length 2, there is a variable in X and a variable in Y. Finally, for each path of length at least 2 with intersection constraints, there is a variable in W. The variables in $X \cup Y \cup Z$ have to be assigned to vertices and the variables in W have to be assigned to paths of length at least 2. Apart from the length constraints (specified as variable types), the endpoints of solution paths need to satisfy the adjacency relationship with respect to T so that we get the desired cycles in G. These constraints are captured using functions $\varGamma$, $\varLambda_1$, $\varLambda_2$, $\varOmega$ from the variables to a collection of subsets of $V(H)$. Furthermore, we have to make additional guesses concerning orientations of paths with respect to the clique partition $\mathcal{Q}$ of H. For example, if we need to find a path between $t_1 \in T$ and $t_2 \in T$ with exactly 2 internal vertices both of which are from H, then we have a variable $x \in X$ and a variable $y \in Y$ corresponding to this constraint. Further, $\varGamma(x)$ is $N(t_1) \cap V(H)$ and $\varGamma(y)$ is $N(t_2) \cap V(H)$. We also require the vertex assigned to x to be adjacent to the vertex assigned to y. To encode this constraint, we set $\varOmega(x) = y$ to imply that in any valid assignment g of vertices to $X \cup Y$, $g(x)$ and $g(\varOmega(x))$ are adjacent. Similarly, if we need to find a path between $t_1 \in T$ and $t_2 \in T$ with exactly 1 internal vertex which is from H, then we have a variable $z \in Z$ and set $\varGamma(z)$ to be $N(t_1) \cap N(t_2) \cap V(H)$. Finally, if we seek a path between $t_1 \in T$ and $t_2 \in T$ with at least 2 internal vertices all of which are from H, then we have a variable $w \in W$ corresponding to this constraint. Further, $\varLambda_1(w)$ is $N(t_1) \cap V(H)$ and $\varLambda_2(w)$ is $N(t_2) \cap V(H)$. The interpretation is that the path assigned to w has to *start* at a vertex in $\varLambda_1(w)$ and *end* at a vertex in $\varLambda_2(w)$. Note that the notion of start and end vertices of a path are derived from a proper interval ordering π of H. It might be the case that w can only be assigned to a path that starts at a vertex in $\varLambda_2(w)$ and ends at a vertex in $\varLambda_1(w)$. To handle this technicality, we create multiple instances of Constrained Path Assignment considering all such possibilities. Thus, by using functions $\varGamma$, $\varOmega$, $\varLambda_1$ and $\varLambda_2$, we ensure that the paths obtained by solving the Constrained Path Assignment instance indeed form cycles (when combined with T) with the required properties specified by the constraints enforced on a solution for $\mathcal{I}$.

Constrained Path Assignment is an optimization problem with an objective to find an assignment (expressed as a pair (h, g) of functions) to W, X, Y and Z such that the maximum number of vertex disjoint triangles in a specific subgraph of H is maximized. Let paths(H) denote the set of paths of length at least 2 that have at most two vertices from Q_i for each $i \in [q]$ (individually). The function g assigns $X \cup Y \cup Z$ to vertices of H such that the assignment restricted to $X \cup Y$ is a valid assignment of length 2 paths. The function h is an assignment of the variables in W to paths in paths(H) satisfying certain intersection constraints propagated from the additional constraints on a solution for $\mathcal{I}$. As we only need to find a nice set of cycles, it suffices to assign each variable in W to a path from this set. Further, we do require h and g to be injective as our interest is in finding vertex-disjoint paths. We also need the images of g and h to be disjoint in the sense that for each pair of elements $w \in W$, $s \in X \cup Y \cup Z$, $g(s) \notin V(h(w))$.

Constrained Path Assignment **Parameter:** $|X| + |Y| + |Z| + |W|$
Instance: A proper interval graph H with clique partition $\mathcal{Q}$ and proper interval ordering π, sets X, Y, Z, W of variables, functions $\varGamma : X \cup Y \cup Z \to \mathcal{R}$, $\varLambda_1 : W \to \mathcal{S}$, $\varLambda_2 : W \to \mathcal{T}$ where $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ are collections of subsets of $V(H)$ and a bijection $\varOmega : X \to Y$.
Feasible Solution: A pair (h, g) of injective functions $h : W \to \text{paths}(H)$ and $g : X \cup Y \cup Z \to V(H)$ with disjoint images such that
(i) For each $s \in X \cup Y \cup Z$, $g(s) \in \varGamma(s)$; for each $x \in X$, $g(x)g(\varOmega(x)) \in E(H)$.
(ii) For each $w \in W$, $h(w)$ is a path between $u \in \varLambda_1(w)$ and $v \in \varLambda_2(w)$.
(iii) For each pair of distinct variables $w, w' \in W$, $V(h(w)) \cap V(h(w')) = \emptyset$.
(iv) For each $i \in [q]$, $|\{w \in W : V(h(w)) \cap Q_i \neq \emptyset\}| \leq 14$.
Optimum Solution: Feasible solution (h, g) that maximizes the number of vertex-disjoint triangles in $H - (V(\text{img}(h)) \cup \text{img}(g))$.

For an instance $\mathcal{J}$ of Constrained Path Assignment with (h, g) as a feasible solution, the *value* of (h, g), denoted by $val_{\mathcal{J}}((h, g))$, is defined as the maximum number of vertex-disjoint triangles in $H - (V(\text{img}(h)) \cup \text{img}(g))$. The value $val_{\mathcal{J}}((h, g))$ of an optimum solution (h, g) is denoted by $opt(\mathcal{J})$. We omit the subscript in the notation for value of an optimum solution for an instance of Constrained Path Assignment if the instance under consideration is implicit. Thus, we reduce the problem of finding cycles in G to the task of finding a collection of vertex-disjoint paths that satisfies certain constraints in H. This completes the first phase of the algorithm.

Lemma 2. *There is an algorithm that, given an instance $\mathcal{I} = (G, T, r)$ of* Cycle Packing*, runs in $\mathcal{O}^*(2^{\mathcal{O}(|T| \log |T|)})$ time and returns a set of $2^{\mathcal{O}(|T| \log |T|)}$ instances of* Constrained Path Assignment *such that $\mathcal{I}$ is a yes-instance if and only if at least one of the returned instances $\mathcal{J}$ satisfies $opt(\mathcal{J}) \geq r - \ell$ for some $\ell \leq |T|$. Further, the parameter of each of the returned instances is a linear function of the parameter of $\mathcal{I}$.*

Phase 2. Consider an instance $\mathcal{J} = (H, \mathcal{Q}, \pi, X, Y, Z, W, \varGamma, \varLambda_1, \varLambda_2, \varOmega)$ of Constrained Path Assignment. Let $X = \{x_1, \ldots, x_{r_X}\}$, $Y = \{y_1, \ldots, y_{r_Y}\}$, $Z = \{z_1, \ldots, z_{r_Z}\}$ and $W = \{w_1, \ldots, w_{r_W}\}$. Recall that each variable in $X \cup Y \cup Z$ has to be assigned to a vertex of H and each variable in W has to be assigned to a path of length at least 2 in H. We color the vertices of H uniformly at random from the color set $[\hat{r}]$ where $\hat{r} = r_X + r_Y + r_Z$. Let $\chi : V(H) \to [\hat{r}]$ denote this coloring.

Definition 2 (Colorful solution). *A feasible solution (h, g) of $\mathcal{J}$ is a* colorful solution *if for any two distinct variables $s, t \in X \cup Y \cup Z$, $\chi(g(s)) \neq \chi(g(t))$.*

Observe that the characteristic of a solution being colorful does not depend on the assignment to the variables in W. We define an *optimum colorful solution* of $\mathcal{J}$ as a colorful solution maximizing $val_{\mathcal{J}}((h, g))$ over all colorful solutions (h, g). We focus on finding an optimum colorful solution of $\mathcal{J}$. Having reduced our search space from the set of all feasible solutions to the set of all colorful solutions, we simplify the instance accordingly. For each $i \in [\hat{r}]$, let V_i denote the set $\{v \in V(H) \mid \chi(v) = i\}$ of vertices of H that were colored i by χ. Let δ be a permutation of $[\hat{r}]$. We use δ to specify the exact color of the vertex that is to be assigned to each $s \in X \cup Y \cup Z$. Define $\widehat{\Gamma} : X \cup Y \cup Z \rightarrow 2^{V(H)}$ as follows: for each $i \in [r_X]$, $\widehat{\Gamma}(x_i) = \Gamma(x_i) \cap V_{\delta(i)}$; for each $i \in [r_Y]$, $\widehat{\Gamma}(y_i) = \Gamma(y_i) \cap V_{\delta(r_X+i)}$; for each $i \in [r_Z]$, $\widehat{\Gamma}(z_i) = \Gamma(z_i) \cap V_{\delta(r_X+r_Y+i)}$. For example, if δ specifies that a variable s is to be assigned to a vertex that has color i, then we restrict the set of vertices that can possibly be assigned to s to those that are colored i. Let $\mathcal{J}(\chi, \delta)$ denote the instance $(H, \mathcal{Q}, \pi, X, Y, Z, W, \widehat{\Gamma}, \Lambda_1, \Lambda_2, \Omega)$ of Constrained Path Assignment. Observe that for each pair of distinct variables $s, t \in X \cup Y \cup Z$, we have $\widehat{\Gamma}(s) \cap \widehat{\Gamma}(t) = \emptyset$.

Observation 1. *Any feasible solution (h, g) of $\mathcal{J}(\chi, \delta)$ is also a colorful solution. Further, if (h, g) is a colorful solution of $\mathcal{J}$, then there exists a permutation δ of $[\hat{r}]$ such that (h, g) is a feasible solution of $\mathcal{J}(\chi, \delta)$.*

For the sake of clarity, we subsequently call the Constrained Path Assignment problem in which, for each pair of variables $s, t \in X \cup Y \cup Z$, $\Gamma(s) \cap \Gamma(t) = \emptyset$ holds, as the Colorful Constrained Path Assignment problem. The standard technique of derandomization of color coding based algorithms [1,9,32] leads to the following result.

Lemma 3. *There is an algorithm that, given an instance $\mathcal{J}$ of* Constrained Path Assignment*, runs in $\mathcal{O}^*(2^{\mathcal{O}(\hat{r} \log \hat{r})})$ time where $\hat{r} = |X| + |Y| + |Z|$ and returns a set of at most $\mathcal{O}^*(\hat{r}! e^{\hat{r}} \hat{r}^{\mathcal{O}(\log \hat{r})})$ instances of* Colorful Constrained Path Assignment *such that at least one of the returned instances $\widehat{\mathcal{J}}$ satisfies $opt(\mathcal{J}) = opt(\widehat{\mathcal{J}})$.*

3 Phase 3: Solving Colorful Constrained Path Assignment

Now, we describe an algorithm that uses a dynamic programming routine and a greedy strategy to solve Colorful Constrained Path Assignment in $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$ time where $k = |X| + |Y| + |Z| + |W|$. Consider an instance $\mathcal{J} = (G, \mathcal{Q}, \pi, X, Y, Z, W, \Gamma, \Lambda_1, \Lambda_2, \Omega)$. Let $\mathcal{Q}$ be the given clique partition $\{Q_1, \ldots, Q_q\}$ of G. First, we observe some properties of a feasible solution which follow from the structure of $\mathcal{Q}$.

Observation 2. *If (h, g) is a feasible solution of $\mathcal{J}$, then there is another feasible solution (h', g) such that $val((h, g)) \leq val((h', g))$ and for each variable $w \in W$, the paths $h'(w)$ and $h(w)$ have the same set of endpoints and $V(h'(w)) \subseteq V(h(w))$. Further, for each variable $w \in W$ and for each $i, j \in [q]$ with $i < j$, every vertex in $V(h'(w)) \cap Q_i$ occurs before any vertex in $V(h'(w)) \cap Q_j$ in $h'(w)$.*

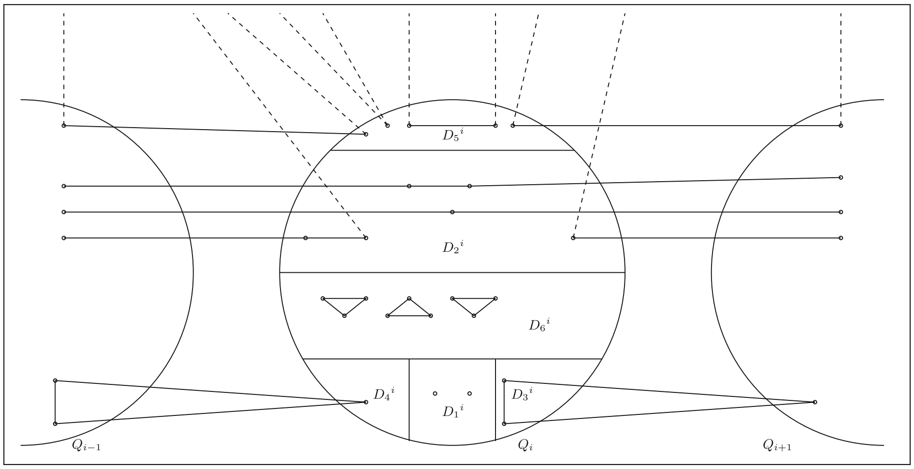


Fig. 1. Illustration of the partition of Q_i into sets D^i_j for each $j \in [6]$

We next define (in Definition 3) the notion of a feasible solution with more special properties. Let (h^*, g^*) be a feasible solution of $\mathcal{J}$. Let $\mathcal{T}$ be a maximum size set of vertex-disjoint triangles in $G - (V(\mathrm{img}(h^*)) \cup \mathrm{img}(g^*))$. Consider the clique Q_i for some $i \in [q]$. Then, Q_i can be partitioned into the following subsets. (1) D^i_1 - the set of vertices of Q_i that are neither in $V(\mathcal{T})$ nor in a path assigned to some variable in W nor assigned to any variable in $X \cup Y \cup Z$. (2) D^i_2 is the set of vertices of Q_i in a path assigned to a variable in W. (3) D^i_3 is the set of vertices of Q_i present in triangles having vertices from both Q_i and Q_{i+1}. (4) D^i_4 is the set of vertices of Q_i present in triangles having vertices from both Q_i and Q_{i-1}. (5) D^i_5 is the set of vertices of Q_i that are assigned to variables in $X \cup Y \cup Z$. (6) D^i_6 is the set of vertices of Q_i that are present in triangles contained in Q_i. An example is given in Fig. 1.

Clearly, $|D^i_2| \le 28$ and by the property of π and $\mathcal{Q}$, it suffices to assume that $|D^i_1|, |D^i_3|, |D^i_4| \le 2$. Consider a pair u, v of vertices such that $u \in D^i_5$ and $v \in D^i_6$. Let $s \in X \cup Y \cup Z$ be the variable such that $g^*(s) = u$. Then, $u \in \Gamma(s)$. Suppose $v \in \Gamma(s)$. Let $(h^*, g^\star)$ denote the pair of functions obtained from (h^*, g^*) by setting $g^\star(s) = v$. Consider the following cases. If $s \in Z$, then $(h^*, g^\star)$ is also a feasible solution of $\mathcal{J}$. Suppose $s \in X \cup Y$ and $g^*(\Omega(s)) \in Q_i \cup Q_{i+1}$. Then, if $\pi(v) > \pi(u)$, then $(h^*, g^\star)$ is also a feasible solution of $\mathcal{J}$ since v and $g^\star(\Omega(s))$ are adjacent by Proposition 1. Similarly, if $s \in X \cup Y$, $g^*(\Omega(s)) \in Q_{i-1}$ and $\pi(v) < \pi(u)$, then $(h^*, g^\star)$ is also a feasible solution of $\mathcal{J}$ since v and $g^\star(\Omega(s))$ are adjacent. Informally, in each of the cases, we obtain an *aligned* (to the left or to the right with respect to π) solution $(h^*, g^\star)$ with $val((h^*, g^*)) = val((h^*, g^\star))$. This leads us to the notion of a *canonical solution* of $\mathcal{J}$. Observe that for any feasible solution (h', g') of $\mathcal{J}$, there is no variable $t \in X \cup Y \cup Z$ distinct from s with $g'(t) = v$. This crucially uses the fact that for each pair of distinct variables $s, t \in X \cup Y \cup Z$, we have $\Gamma(s) \cap \Gamma(t) = \emptyset$ – a property achieved by color coding.

Definition 3 (Canonical solution). *Let (h, g) be a feasible solution of $\mathcal{J}$. Let $\mathcal{T}$ be a maximum size set of vertex-disjoint triangles in $G - (V(\text{img}(h)) \cup \text{img}(g))$. Let $\mathcal{T}'$ denote the set of triangles in $\mathcal{T}$ that are contained in Q_i for some $i \in [q]$. Let $V' = V(\mathcal{T}') \cup \text{img}(g)$ and $D_i = V' \cap Q_i$. (h, g) is a* canonical solution *if*

- *For each $i \in [q]$ and $s \in X \cup Y$ with $g(s) \in Q_i$ and $g(\Omega(s)) \in Q_{i+1} \cup Q_i$, $g(s)$ is the vertex that maximizes $\pi(g(s))$ over all vertices in $D_i \cap \Gamma(s)$.*
- *For each $i \in [q]$ and $s \in X \cup Y$ with $g(s) \in Q_i$ and $g(\Omega(s)) \in Q_{i-1}$, $g(s)$ is the vertex that minimizes $\pi(g(s))$ over all vertices in $D_i \cap \Gamma(s)$.*
- *For each $i \in [q]$ and $s \in Z$ with $g(s) \in Q_i$, $g(s)$ is the vertex that maximizes $\pi(g(s))$ over all vertices in $D_i \cap \Gamma(s)$.*

Note that the function h plays no role in deciding if (h, g) is indeed a canonical solution or not. Next, we prove the existence of canonical solutions.

Lemma 4. *If (h, g) is a feasible solution of $\mathcal{J}$, then there is a canonical solution (h, g^*) of $\mathcal{J}$ with $val((h, g)) = val((h, g^*))$.*

Proof. Let $Q_0 = \emptyset$. For an integer $i \in [q] \cup \{0\}$, we say that (h, g) is an *i-canonical solution* if it satisfies the properties in Definition 3 for cliques the $Q_0, Q_1, \ldots, Q_i$. In this context, a q-canonical solution is a canonical solution. We show the existence of a q-canonical solution by induction on q. For $q = 0$, observe that (h, g) is a q-canonical solution. Suppose (h, g) is an $(i-1)$-canonical solution for some $i \geq 1$. We show that there is another feasible solution (h, g^*) that is an i-canonical solution. Let $\mathcal{T}$ be a maximum size set of vertex-disjoint triangles in $G - (V(\text{img}(h)) \cup \text{img}(g))$. Initialize (h, g^*) to (h, g). Let V' denote the vertices of Q_i that are present in triangles that are completely contained in some clique Q_j, $j \in [q]$. Let $D_i = V' \cap Q_i$.

Let s be a variable in $X \cup Y$ such that $g(s) \in Q_i$. Then, either $g(\Omega(s)) \in Q_{i-1}$ or $g(\Omega(s)) \in Q_{i+1} \cup Q_i$. In the former case, update $g^*(s)$ to the vertex v in $D_i \cap \Gamma(s)$ that minimizes $\pi(v)$. From Proposition 2, as $\pi(g^*(s)) \leq \pi(g(s))$, $\pi(g(\Omega(s))) < \pi(g^*(s))$ and $(g(s), g(\Omega(s))) \in E(G)$, we have $(g^*(s), g^*(\Omega(s))) \in E(G)$. In the latter case, update $g^*(s)$ to the vertex v in $D_i \cap \Gamma(s)$ that maximizes $\pi(v)$. From Proposition 2, as $\pi(g^*(s)) \geq \pi(g(s))$, $\pi(g(\Omega(s))) > \pi(g^*(s))$ and $(g(s), g(\Omega(s))) \in E(G)$, we have $(g^*(s), g^*(\Omega(s))) \in E(G)$. In any case, v is either $g(s)$ or v is in a triangle T of $\mathcal{T}$ with vertices only from Q_i. Therefore, the set $\mathcal{T}^*$ of triangles obtained from $\mathcal{T}$ from replacing v by $g(s)$ is a set of same size as $\mathcal{T}$. Execute this replacement procedure for all variables in $X \cup Y$ that are assigned to vertices in Q_i by g. At the end of this reassignment, we have the desired i-canonical solution. □

By Lemma 4, it suffices to find an optimum solution of $\mathcal{J}$ that is canonical. We describe a dynamic programming algorithm finding such a solution (if one exists) by processing the cliques in $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ in the increasing order of their indices. Let $Q_{q+1} = \emptyset$. For each $i \in [q+1]$, let G_i be the subgraph of G induced by $\bigcup_{j=1}^{i} Q_j$. For each $i \in [q+1]$, we maintain a table $\mathbb{T}_i$. Before describing the entries in $\mathbb{T}_i$, we give an overview of what we would ideally like them to store. Let (h^*, g^*) be an optimum canonical solution of $\mathcal{J}$. Let $\mathcal{T}$ be a maximum size set of vertex-disjoint triangles in $G - (V(\text{img}(h^*)) \cup \text{img}(g^*))$. As each of the graphs in the sequence $G_1, G_2, \ldots, G_q, G_{q+1}$ is a subgraph of the graph succeeding it, we like to process the cliques $Q_1, \ldots, Q_q, Q_{q+1}$ from "left-to-right", that is, from Q_1 to Q_{q+1}, in order to compute (h^*, g^*) (or a feasible solution $(h^\star, g^\star)$ with $val((h^\star, g^\star)) = val((h^*, g^*))$). We would first like to understand how (h^*, g^*) and $\mathcal{T}$ look like when they are restricted to G_i for a fixed $i \in [q+1]$. This could shed insight into the subproblem that we want to solve on G_i and the (partial) solution that we want to store for G_i. Ideally, we want to store the number (or set) of triangles in $\mathcal{T}$ that are contained in G_i. Let us call this set $\mathcal{T}'$. Any triangle T in $\mathcal{T} \setminus \mathcal{T}'$ has a vertex from Q_j for some $j \geq i+1$. We cannot *see* this triangle until we look at G_j. Our subproblem on G_i can afford to forget such triangles provided it remembers the vertices from Q_i that are present in these triangles. That is, we need to remember the set $Next^*$ of vertices in Q_i that are present in triangles in $\mathcal{T}$ that have a vertex from both Q_i and Q_{i+1}. Note that no vertex in Q_i can be in a triangle that has a vertex from Q_{i+2} by Proposition 2. Let Not^* be the set of vertices of Q_i that are not in the set $V(\mathcal{T}) \cup V(\text{img}(h^*)) \cup \text{img}(g^*)$. Let us partition $\mathcal{T}'$ into $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\mathcal{T}_1$ is the set of triangles contained in G_{i-1} and $\mathcal{T}_2 = \mathcal{T}' \setminus \mathcal{T}_1$. Suppose we have computed $|\mathcal{T}_1|$ by solving the subproblem on G_{i-1}. Then, $|\mathcal{T}_1|$ can be used to compute $|\mathcal{T}'|$ provided we know the set $Q_i \cap V(\mathcal{T}_2)$. That is, we need to know the set $Prev^*$ of vertices in Q_i that are present in triangles in $\mathcal{T}'$ that have vertices from both Q_i and Q_{i-1}. So far, we have identified two types of vertices (those in $Prev^*$ and those in $Next^*$) in Q_i with respect to (h^*, g^*). In order to understand the role of other vertices in Q_i, we partition Q_i into 6 sets D_1^i to D_6^i as defined earlier.

Note that $D_1^i = Not^*$, $D_3^i = Next^*$ and $D_4^i = Prev^*$. When we process Q_i (to solve the problem on G_i), we guess D_j^i for each $j \in [4]$. The number of such guesses is bounded by a polynomial (in $|Q_i|$). We guess (at most 2^k choices) the set $S_i^* \subseteq X \cup Y \cup Z$ of variables assigned to vertices from Q_i by g^*. Similarly, we guess the set $L_i^* \subseteq W$ of variables assigned to paths containing a vertex from Q_i by h^*. We also map these variables to vertices in D_2^i as there are only a polynomial number of choices. Further, we guess the set $L_{new}^* \subseteq L_i^*$ of variables assigned to paths that start at a vertex in Q_i by h^*. Using the properties of a canonical solution, we greedily map the variables in S_i^* to vertices in $Q_i \setminus (D_1^i \cup D_2^i \cup D_3^i \cup D_4^i)$. These vertices form the set D_5^i. Each vertex in D_6^i is in a triangle contained in Q_i. As Q_i is a clique, once D_5^i is determined, any set of $(|Q_i| - \sum_{j=1}^{5} |D_j^i|)/3$ triangles consisting of the remaining vertices is good enough for our solution. To solve the subproblem on G_i using the previously computed solution on G_{i-1}, we need information regarding the solution contained in G_{i-1}. Let S^* be the set of variables in $X \cup Y \cup Z$ assigned to vertices in G_{i-1} by g^* and L^* be the set of variables in W assigned to paths in G_{i-1} by h^*. As the number of choices for S^* and L^* is $\mathcal{O}(2^k)$, we can guess S^* and L^* while processing Q_i. In $\mathbb{T}_i$, for each choice of $Prev, Next, Not \subseteq Q_i$, $S, S_i \subseteq X \cup Y \cup Z$, $L, L_i, L_{new} \subseteq W$, we store the possibility of the existence of an optimum solution

(h^*, g^*) to $\mathcal{J}$ such that $S = S^*$, $L = L^*$, $S_i = S_i^*$ and $L_i = L_i^*$. Further, for each $w \in L_i$, $h_1(w)$ is the first vertex in $h^*(w)$ that is from Q_i and $h_2(w)$ is the second vertex (if it exists) in $h^*(w)$ that is from Q_i. Moreover, $L_{new} = L^*_{new}$. $\mathbb{T}_i$ stores the information relating to a maximum size set $\mathcal{T}$ of vertex-disjoint triangles in $G - (V(\text{img}(h^*)) \cup \text{img}(g^*))$ such that $Not = Not^*$ and $|Q_i| - (|Prev| + |Next| + |Not| + |\text{img}(h_1)| + |\text{img}(h_2)| + |S_i|)/3$ is the number of triangles in $\mathcal{T}$ that are contained in Q_i. Also, for each $v \in Prev$, there is a triangle in $\mathcal{T}$ containing v and a vertex from Q_{i-1}. Finally, for each $v \in Next$, there is a triangle in $\mathcal{T}$ containing v and a vertex from Q_{i+1}. Computing the tables $\mathbb{T}_i$ (of size $\mathcal{O}^*(2^k)$) in the increasing order of i leads to the following.

Lemma 5. *There is an algorithm that solves* Colorful Constrained Path Assignment *in* $\mathcal{O}^*(2^{\mathcal{O}(|X|+|Y|+|Z|+|W|)})$ *time.*

Combining Lemma 5 with the reductions described establishes Theorem 1.

4 Concluding Remarks

We described an FPT algorithm for Cycle Packing parameterized by the size of a proper interval deletion set. The starting point of our algorithm is identifying the structure of cycles in the input graph. Using this structure, we transformed the problem of finding cycles into the problem of finding paths in the proper interval subgraph. Our approach essentially consists of a guessing phase, a coloring phase and a dynamic programming routine using a greedy strategy. Our belief is that this approach is quite general and can be useful in solving many other problems with the same parameterization. Exploring the applicability of the same for other problems is a natural direction of research.

As mentioned earlier, Cycle Packing parameterized by the size of a proper interval vertex deletion set does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Recently, a kernelization framework (called *lossy kernelization*) that is less stringent than the notion of polynomial kernels was introduced in [29]. It was shown that there are many problems (including Cycle Packing parameterized by the solution size) without classical polynomial kernels that admit lossy polynomial kernels. It is interesting to explore the possibility of such a kernel for our problem. Finally, the status of Cycle Packing on interval graphs is an inevitable line of study.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM **42**(4), 844–856 (1995)
2. Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Comput. Sci. **5**(1), 59–68 (1994)
3. Bodlaender, H.L., Jansen, B.M.P.: Vertex cover kernelization revisited: upper and lower bounds for a refined parameter. Theory Comput. Syst. **63**(2), 263–299 (2013)
4. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Kernel bounds for path and cycle problems. Theor. Comput. Sci. **511**, 117–136 (2013)

5. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. Theor. Comput. Sci. **412**(35), 4570–4578 (2011)
6. Cai, L.: Parameterized complexity of vertex colouring. Discrete Appl. Math. **127**(3), 415–429 (2003)
7. Cao, Y.: Unit interval editing is fixed-parameter tractable. Inf. Comput. **253**(Part 1), 109–126 (2017)
8. Cao, Y., Marx, D.: Interval deletion is fixed-parameter tractable. ACM Trans. Algorithms **11**(3), 21:1–21:35 (2015)
9. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-21275-3
10. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 150–159 (2011)
11. Diestel, R.: Graph Theory. GTM, vol. 173. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-53622-3
12. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer, London (2013). https://doi.org/10.1007/978-1-4471-5559-1
13. Erdös, P., Pósa, L.: On independent circuits contained in a graph. Can. J. Math. **17**, 347–352 (1965)
14. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. J. ACM **35**(3), 727–739 (1988)
15. Fellows, M.R., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F.A., Saurabh, S.: The complexity ecology of parameters: an illustration using bounded max leaf number. Theory Comput. Syst. **45**(4), 822–848 (2009)
16. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-29953-X
17. Fomin, F.V., Saurabh, S., Villanger, Y.: A polynomial kernel for proper interval vertex deletion. SIAM J. Discrete Math. **27**(4), 1964–1976 (2013)
18. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co., Amsterdam (2004)
19. Guruswami, V., Pandu Rangan, C., Chang, M.S., Chang, G.J., Wong, C.K.: The K_r-packing problem. Computing **66**(1), 79–89 (2001)
20. Gutin, G., Kim, E.J., Lampis, M., Mitsou, V.: Vertex cover problem parameterized above and below tight bounds. Theory Comput. Syst. **48**(2), 402–410 (2011)
21. Gutin, G., Yeo, A.: Constraint satisfaction problems parameterized above or below tight bounds: a survey. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond. LNCS, vol. 7370, pp. 257–286. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8_14
22. Jansen, B.M.P.: The power of data reduction: kernels for fundamental graph problems. Ph.D. thesis, Utrecht University, The Netherlands (2013)
23. Jansen, B.M.P., Fellows, M.R., Rosamond, F.A.: Towards fully multivariate algorithmics: parameter ecology and the deconstruction of computational complexity. Eur. J. Comb. **34**(3), 541–566 (2013)
24. Jansen, B.M.P., Raman, V., Vatshelle, M.: Parameter ecology for feedback vertex set. Tsinghua Sci. Technol. **19**(4), 387–409 (2014)
25. Ke, Y., Cao, Y., Ouyang, X., Wang, J.: Unit interval vertex deletion: fewer vertices are relevant (2016). arXiv:arXiv:1607.01162

26. Kloks, T.: Packing interval graphs with vertex-disjoint triangles. CoRR, abs/1202.1041 (2012)
27. Lokshtanov, D., Mouawad, A., Saurabh, S., Zehavi, M.: Packing cycles faster than Erdös-Pósa. In: 44th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 71:1–71:15 (2017)
28. Lokshtanov, D., Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: Faster parameterized algorithms using linear programming. ACM Trans. Algorithms **11**(2), 15:1–15:31 (2014)
29. Lokshtanov, D., Panolan, F., Ramanujan, M.S., Saurabh, S.: Lossy kernelization. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pp. 224–237 (2017)
30. Looges, P.J., Olariu, S.: Optimal greedy algorithms for indifference graphs. Comput. Math. Appli. **25**(7), 15–25 (1993)
31. Manić, G., Wakabayashi, Y.: Packing triangles in low degree graphs and indifference graphs. Discrete Math. **308**(8), 1455–1471 (2008)
32. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: IEEE 36th Annual Symposium on Foundations of Computer Science (FOCS), pp. 182–191 (1995)
33. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Comb. Theory Ser. B **63**(1), 65–110 (1995)
34. van Bevern, R., Komusiewicz, C., Moser, H., Niedermeier, R.: Measuring indifference: unit interval vertex deletion. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 232–243. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16926-7_22
35. van't Hof, P., Villanger, Y.: Proper interval vertex deletion. Algorithmica **65**(4), 845–867 (2013)

Satisfying Neighbor Preferences on a Circle

Danny Krizanc[1], Manuel Lafond[2], Lata Narayanan[3], Jaroslav Opatrny[3(✉)], and Sunil Shende[4]

[1] Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT, USA
dkrizanc@wesleyan.edu

[2] Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada
mlafond2@uOttawa.ca.com

[3] Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, Canada
{lata,opatrny}@cs.concordia.ca

[4] Department of Computer Science, Rutgers University, Camden, USA
sunil.shende@rutgers.edu

Abstract. We study the problem of satisfying seating preferences on a circle. We assume we are given a collection of n agents to be arranged on a circle. Each agent is colored either blue or red, and there are exactly b blue agents and r red agents. The w-neighborhood of an agent A is the sequence of $2w + 1$ agents at distance $\leq w$ from A in the clockwise circular ordering. Agents have preferences for the colors of other agents in their w-neighborhood. We consider three ways in which agents can express their preferences: each agent can specify (1) a preference list: the *sequence* of colors of agents in the neighborhood, (2) a preference type: the *exact* number of neighbors of its own color in its neighborhood, or (3) a preference threshold: the *minimum* number of agents of its own color in its neighborhood. Our main result is that satisfying seating preferences is fixed-parameter tractable (FPT) with respect to parameter w for preference types and thresholds, while it can be solved in $O(n)$ time for preference lists. For some cases of preference types and thresholds, we give $O(n)$ algorithms whose running time is independent of w.

Keywords: Seating arrangement · Linear algorithm · FPT algorithm

1 Introduction

Alice has invited a large group of people to a dinner party, just before a most contentious election between the Red and Blue political parties. The guests are to be seated at a large circular table. She has now started receiving urgent requests from her guests about the seating arrangements. Some guests want to be assured that they won't be seated close to anyone from the other party. Other guests are more tolerant; they would be happy so long as a majority of their neighbors belong to their own party. There are also argumentative guests who insist on being seated near people mostly from the other party; they want to have a chance to argue with their neighbors all evening. Can Alice satisfy all the guests' preferences? How can she figure out a seating arrangement acceptable to all her guests?

In this paper, we study the problem of satisfying seating preferences on a circle. We assume we are given a collection of n *agents* to be arranged on a circle. Each agent is colored either blue (B) or red (R), and there are exactly b blue agents and $r = n - b$ red agents in all. Any specific clockwise ordering of these n colored agents around the circle is called a *configuration* or *n-configuration*; we are interested in exploring questions about the existence of configurations that, for each agent, satisfy a given constraint on the colors of agents in its neighborhood. For a fixed value w, called *half-window size*, $1 \leq w \leq (n-1)/2$, we define the neighborhood of an agent as the w agents preceding it, the agent itself, and the w agents following it, in the clockwise ordering. The agents preceding and following it are called the *proper neighbors* of the agent.

Agents can express **preferences** for the colors of agents in their neighborhood in several possible ways. We consider three ways in which color specifications can be expressed. For every agent, we specify its:

Preference list: the exact sequence of colors desired in its neighborhood. Each preference list is a string over $\{B, R\}$ of length $2w + 1$.

Preference type: the desired *number* of proper neighbors with the *same color* as the agent. Furthermore, the number of red (resp. blue) nodes with preference type i is denoted r_i (resp. b_i).

Preference threshold: the *minimum number* of proper neighbors with the *same color* as the agent. Furthermore, the number of red (resp. blue) nodes with threshold i is denoted ρ_i (resp. β_i).

We assume that *all agents specify their preferences in the same manner*, e.g., they all specify (possibly different) preference thresholds. For any one of the three preference specification methods, a given configuration of agents is said to be **valid** if and only if all the agents have their preferences satisfied in their respective neighborhoods within the configuration. For example, the 10-configuration in Fig. 1 specified by the string $RRBRRRBRRR$, is valid for the following preference specifications with half-window size $w = 2$:

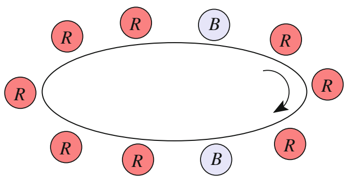


Fig. 1. An example of a seating configuration of agents (Color figure online)

- Preference lists: Two blue agents with lists $RRBRR$; two red agents with lists $RBRRR$; two red agents with lists $RRRBR$; one red agent for lists $BRRRB$, $BRRRR$, $RRRRR$, and $RRRRB$.
- Preference types: Two blue agents with types 0, one red agent with type 2, six red agents with type 3 and one red agent with type 4.
- Preference thresholds: Two blue agents with threshold 0; two red agents with threshold 2 and six red agents with threshold 3. Note that this configuration also satisfies many other preference thresholds, for example, two blue agents with threshold 0, and eight red agents with threshold 1.

We are interested in the following problem: *For a given preference specification with n agents with a given half-window size w, construct a valid n-configuration, or determine that it does not exist.*

In the sequel, we consider the above problem for each of the three kinds of preference specifications. We consider the cases when all nodes have the same preference (homogeneous preferences), when the nodes of the same color have the same preference (homogeneous within type), and the general (heterogeneous preference) case.

1.1 Related Work

Our problem was largely motivated by the influential work of Schelling [19,20] on how the preferences of individuals can potentially lead to the undesirable effect of global segregation. In the Schelling's model, we imagine people of two different types (Red and Blue in our example) who are placed on a line (or later on a grid). Each person would like at least a fraction τ of their size w neighborhood to be of their type. Consider a process whereby a randomly chosen pair of people (of different type) who are both unhappy in their current location are allowed to switch positions. Schelling showed via simulations that this would eventually lead to segregation by type even in the case where $\tau \leq \frac{1}{2}$. His observations led to a great deal of research in both the sociological and mathematics literature in attempts to verify his conclusions [2,3,8,10,16,21,22]. Recently, this analysis has been taken up in the theoretical computer science literature [4,11]. In that work, the model is analyzed as a random process and it is shown that the expected size of the resulting segregated neighborhoods is polynomial in w on the line [4] and exponential in w on the grid [11] but that in both cases it is independent of the overall number of participants.

Our work may be thought of as a first attempt to analyze this model from a worst-case deterministic perspective rather than on average. Given the preferences of the individuals, we ask whether there is a configuration under which they may all be satisfied. As far as we can determine, we are the first to consider this question. However, as might be expected, related problems have been studied in the graph coloring literature. In particular, a *perfect 2-coloring* of a graph is a 2-coloring of the graph where nodes of a given color are required to have a given number of neighbors of the same color. Parshina [17] characterized the perfect 2-colorings of infinite circulant graphs with a continuous set of distances which corresponds to the infinite line with each node connected to all of its neighbors within a fixed distance. We rely upon this characterization in order to solve the version of our problem when all nodes of the same color have the same preference type (see Sect. 4.1). Somewhat less related but of a similar flavor are (m, k)-*defective colorings* whereby the nodes of a graph are colored by m colors in such a way that each vertex is adjacent to at most k vertices of the same color. This concept was introduced in [1,5] and continues to be studied [6,7,13]. Since we are interested in colorings with at least some number of vertices of the same color in each vertex's neighborhood, these results do not appear to be directly applicable to our problems.

1.2 Our Results

We assume that the input for a seating arrangement problem is a sequence of preference specifications for all n agents, all of the same kind.

1. For preference lists, we give in Sect. 3 an $O(n)$ algorithm to construct a valid configuration.
2. In Sect. 4.1 we give an algorithm to construct a valid configuration for homogeneous preference types in $O(n)$ time.
3. For heterogeneous preference types and for preference thresholds we show in Sects. 4.2 and 5 that satisfying seating preferences is FPT for parameter w. In particular, for each of the two cases, we give an algorithm to construct a valid configuration that has time complexity $O(n + 2^{2^{2w+1}} w(w2^{2w+1} + \log n)2^{(2w+1)2.5\cdot 2^{2w+1}} \log n)$.

2 Notation

We use strings to represent configurations of agents, taken circularly. Following standard terminology, for any string u, we denote by $|u|$ the length of u and by u^k the string u repeated k times. We denote by u^+ the infinite set of strings consisting of *one or more repetitions* of the string u, and $\overline{u}$ denotes the *complementary* string obtained by replacing each color symbol in u by its complementary color, i.e., replacing B with R and vice-versa. An n-*configuration* is a specific clockwise ordering of n agents around the circle, i.e., a *cyclic string* $s[0]\ s[1]\ \dots\ s[n-1]$, denoted by $\mathbf{s[0:n]}$, where the i^{th} agent has color $s[i] \in \{B, R\}$. For $i < j$, let $s[i:\ j]$ denote the contiguous substring of agents $s[i]\ s[i+1]\ \dots\ s[j-1]$ in the configuration. Thus $s[i:\ i+1] = s[i]$. We follow the convention that sequence indices: (a) always start from reference position 0; (b) are interpreted modulo n; and (c) increase in *clockwise order* around the circle. Thus given a half-window size w, the neighborhood of agent $s[i]$ consists of $s[i-w:\ i]$, its *left* neighbors, $s[i]$ itself, and $s[i+1:\ i+w+1]$, its *right* neighbors. Since we assume that $n \geq 2w+1$, each agent has $2w$ distinct neighbors. A *maximal* contiguous sequence of agents of the same color is called a *run* of that color and is denoted as a string, e.g. B^k is a blue run of length k.

3 Preference Lists

When given the preference lists of agents, the feasibility problem is solvable in polynomial time: this can be done by constructing an appropriate directed graph representation of the preferences, and finding an Euler tour in the graph. The technique was originally proposed for DNA fragment assembly in [18]. Let $S = \{s_0, s_1, \ldots, s_{n-1}\}$ be the set of preference lists given by the n agents (recall that each s_i is a string of length $2w + 1$ over the color alphabet $\{B, R\}$). Define S' to be the set of $2w$-length strings obtained by dropping either the *first* or the *last* symbol from a preference list in S. That is, $s' \in S'$ if and only if there exists a preference list $s \in S$ so that either $s' = s[0 : \ 2w]$ or $s' = s[1 : \ 2w + 1]$.

We now construct a digraph $G = (V, E)$ as follows. We set $V = S'$, so that each vertex is uniquely labeled with a string in S'. Since the i^{th} agent has preference list s_i, we create a directed edge labeled i from the node labeled $s_i[0 : \ 2w]$ to the node labeled $s_i[1 : \ 2w + 1]$. As an example, let $w = 2$, and suppose that the preference list for the i^{th} agent is $s_i = RRBBR$. Then i is the *edge-label* of an edge from the node labeled $RRBB$ to the node labeled $RBBR$. Notice that if there are two agents with the same preference list, then there will be two edges between the same two vertices in the graph G.

Suppose i and j are two consecutive edge-labels along a path in G. Then, notice that agent j can be placed to the right of agent i in a configuration. In the example above, the path

$$RRBB \xrightarrow{i} RBBR \xrightarrow{j} BBRR$$

indicates that agent j with preference list $s_j = RBBRR$ can follow agent i in a configuration. Clearly, there is an Euler tour in the graph G iff there is a valid n-configuration of the agents. Since the graph G has at most $2n$ nodes and exactly n edges, we get:

Theorem 1. *For a preference list specification, a valid configuration, if one exists, can be found in time $O(n)$.*

4 Preference Types

4.1 Homogeneous Preference Types

We first consider the situation where all the blue agents have *homogeneous* preferences for exactly i blue neighbors each and likewise, all the red agents have homogeneous preferences for exactly j red neighbors each. Such configurations were studied by Parshina [17] where some characterizations were provided for *infinite* valid strings. In particular, the paper does not consider the values of n, r and b, the actual number of red and blue agents present.

In what follows, we will consider that w, i and j are fixed. A string s over alphabet $\{R, B\}$ is *valid* if each blue agent in s has i blue neighbors, and each red agent has j red neighbors.

The analysis of homogeneous preference types is divided into two parts. We first paraphrase the results from Parshina [17] that establish various conditions under which valid strings exists, regardless of the values of r and b. Then we use these results to decide, for given values of r and b, if a valid configuration exists.

The following result, stated without proof, is paraphrased from the paper by Parshina [17]. It characterizes the valid strings for a given w, i (the blue preference type) and j (the red preference type).

Lemma 1 **([17]).** *Let s be a string over alphabet $\{R, B\}$. Then s is valid if and only if one of the following statements holds:*

1. *$i + j = 2w - 2$ and*
   - *both i and j are even and $s \in u^+$ for some string u of length $w + 1$ containing $1 + i/2$ Bs and $1 + j/2$ Rs, or*
   - *$i = j$ and $s \in (u\bar{u})^+$ for some string u of length w.*
2. *$i + j = 2w - 1$ and $s \in u^+$ for some string u of length $2w + 1$ containing $i + 1$ Bs and $j + 1$ Rs.*
3. *$i + j = 2w$ and*
   - *both i and j are even and $s \in u^+$ for some string u of length w containing $i/2$ Bs and $j/2$ Rs, or*
   - *$i = j$ and $s \in (u\bar{u})^+$ for some string of length $w + 1$.*

Lemma 1 immediately allows us to construct valid configurations for some values of r, b, n, but not for all values for which a valid configuration exists. In the remainder of this section, we find necessary and sufficient conditions for the existence of all valid configurations. To do this, we need to identify appropriate periods of the valid strings described in Lemma 1.

Characterizing the Periods of Valid Strings. For a string s, we call p a *period* of s if $s \in p^+$. We say that s is *aperiodic* if the only period of s is itself. A period p of s is *minimal* if p itself is aperiodic. We start with the next simple result that allows us to search for "good" minimal periods of valid strings.

Proposition 1. *There exists a valid configuration $s[0:n]$ for given r and b and $n = r + b$ if and only if there exists a valid string $s' \in p^+$ with minimal period p such that $n = k \cdot |p|$ for some integer k, and p has r/k red colors and b/k blue colors.*

Proof. If $s[0:n]$ is valid for r and b, then $s' = s$ along with any minimal period p of s' must satisfy the statement. Conversely, it is clear that p^k as described is a valid configuration for r and b. $\square$

For any integer k, we define $t(k)$ to be the *largest power of two that divides k* (with $t(k) = 1$ if k is odd). We first need the two following intermediate results.

Lemma 2. *Suppose that $i + j = 2w$. Then there exists a valid configuration $s \in (u\overline{u})^+$ such that $|u| = w + 1$ and $u\overline{u}$ has a period p if and only if $i = j$, $|p|$ divides $2w + 2$, $|p|$ is a multiple of $t(2w + 2)$ and $p = q\overline{q}$ for some string q.*

Proof. ($\Rightarrow$) The necessity of $i = j$ is due to Lemma 1(c). The fact that $|p|$ divides $2w + 2$ is immediate, since $|u\overline{u}| = 2w + 2$ and p is a period of $u\overline{u}$. Let k be the integer such that $u\overline{u} = p^k$. We claim that $|p|$ does not divide $w + 1$. Assume on the contrary that $|p|$ also divides $w + 1$, say $|p| = (w + 1)/h$ for some integer h. Then $|p^h p^h| = 2w + 2 = |p^k|$, implying $u\overline{u} = p^h p^h$, which is a contradiction since it implies $u = \overline{u}$.

Given the above claims, suppose now that $|p|$ is not a multiple of $t(2w + 2)$. Then $t(|p|) < t(2w + 2)$. As $|p^k| = 2w + 2$, we have $t(k|p|) = t(2w + 2)$, and so $t(k) > 1$ and k must be even, say $k = 2k'$. But $2k'|p| = 2w + 2$ implies that $k'|p| = w + 1$, a contradiction since $|p|$ does not divide $w + 1$.

Finally to see that $p = q\overline{q}$ for some q, first note that this trivially holds if $p = u\overline{u}$. Otherwise, $|p| < 2w+2$. Write $p = p_1p_2$, where $|p_1| = |p_2| = |p|/2$ (note that given the above, $|p|$ must be even). Since $(u\overline{u})^+ = (p_1p_2)^k$, we either have $u = (p_1p_2)^{k/2}$ or $u = (p_1p_2)^{(k-1)/2}p_1$. The former is not possible, as it would imply $u = \overline{u}$. The latter implies $\overline{u} = p_2(p_1p_2)^{(k-1)/2}$, which in turn implies $p_1 = \overline{p_2}$.

($\Leftarrow$) Let k be such that $|p| = (2w + 2)/k$. Because $|p|$ contains all the even prime factors of $2w+2$ and $|p|$ divides $2w+2$, k must be a product of odd factors and hence must be odd. Write $u = (q\overline{q})^{(k-1)/2}q$ and $\overline{u} = \overline{q}(q\overline{q})^{(k-1)/2}$. One can check that $u\overline{u} = p^k$ and, by Lemma 1, $s \in u^+$ is a valid configuration. □

We now give the analogous statement for the case $i + j = 2w - 2$. Since its proof is essentially identical to the above, it is omitted.

Lemma 3. *Suppose that $i + j = 2w - 2$. Then there exists a valid configuration $s \in (u\overline{u})^+$ such that $|u| = w$ and $u\overline{u}$ has period p if and only if $i = j$, p divides $2w$, $|p|$ is a multiple of $t(2w)$ and $p = q\overline{q}$ for some string q.*

We are now ready to give the characterization of minimal periods of s.

Theorem 2. *There exists a valid configuration $s \in p^+$ with minimal period p if and only if one of the following conditions holds:*

1. *$i + j = 2w - 2$, $|p| = (w + 1)/k$ for some integer k and p has $(i + 2)/(2k)$ B's and $(j + 2)/(2k)$ R's;*
2. *$i + j = 2w - 1$, $|p| = (2w + 1)/k$ for some integer k and p has $(i + 1)/k$ B's and $(j + 1)/k$ R's;*
3. *$i + j = 2w$, $|p| = w/k$ for some integer k, and p has $i/(2k)$ B's and $j/(2k)$ R's;*
4. *$i = j = w$, $|p|$ divides $2w + 2$, $|p|$ is a multiple of $t(2w + 2)$ and $p = q\overline{q}$ for some string q.*
5. *$i = j = w - 1$, $|p|$ divides $2w$, $|p|$ is a multiple of $t(2w)$ and $p = q\overline{q}$ for some string q.*

Proof. Let us first show that any of the conditions of statement is sufficient. For (1) and (3), $s \in p^+$ is valid by Lemma 1 parts 1 and 3, respectively. For (2), $s \in p^+$ is valid by Lemma 1 part 2. Condition (4) is sufficient by Lemma 2 and (5) by Lemma 3.

We now show that one of these conditions must hold for $s \in p^+$ to be valid. Suppose first that $i + j = 2w - 2$. By Lemma 1, either $s \in u^+$ for some u with $|u| = w + 1$, or $s \in (u\overline{u})^+$ for some u with $|u| = w$. If $s \in u^+$ with $|u| = w + 1$, let p be the minimal period of u, with $u = p^k$. Then u must have $i/2 + 1$ B's and $j/2 + 1$ R's, from which it follows that (1) holds. If instead $s \in (u\overline{u})^+$ with $|u| = w$, then by Lemma 3, we have that (5) holds.

Suppose that $i + j = 2w$. Again, we handle the two possible cases prescribed by Lemma 1. If $s \in u^+$ with $|u| = w$, let p be the minimal period of u, with $u = p^k$. Then (3) must hold. If $s \in (u\overline{u})^+$ with $|u| = w + 1$, by Lemma 2, it is (4) that holds.

Finally suppose that $i + j = 2w - 1$. By Lemma 1, if $s \in u^+$ and p is a period of u, $u = p^k$, then p must have $(i + 1)/k$ blue characters and $(j + 1)/k$ red characters, and (2) holds. □

The problem of constructing a configuration therefore amounts to building a string p such that $|p|$ divides $r + b$ and that satisfies one of the above conditions. Proposition 1 combined with Theorem 2 can be used to derive when such a string p can be constructed. It is not hard to check that each case enumerated in the following corollary corresponds to one of the cases of Theorem 2.

Corollary 1. *For given w, i, j, r and b, there exists a valid configuration if and only if one of the following conditions holds:*

1. *$i + j = 2w - 2$, $w + 1$ and $r + b$ have a common divisor d such that $(2w + 2)/d$ divides both $i + 2$ and $j + 2$;*
2. *$i + j = 2w - 1$, $2w + 1$ and $r + b$ have a common divisor d such that $(2w + 1)/d$ divides both $i + 1$ and $j + 1$;*
3. *$i + j = 2w$, w and $r + b$ have a common divisor d such that $2w/d$ divides both i and j;*
4. *$i = j = w$, $2w + 2$ and $r + b$ have a common divisor d such that d is a multiple of $t(2w + 2)$;*
5. *$i = j = w - 1$, $2w$ and $r + b$ have a common divisor d such that d is a multiple of $t(2w)$.*

Theorem 3. *If all red agents have preference type i and all blue agents have preference type j, a valid configuration, if one exists, can be constructed in $O(n)$ time.*

Proof. To determine the existence of a valid configuration, it suffices to find a divisor d that satisfies one of the cases in Corollary 1. As d must be a divisor of one of $\{w, w+1, 2w, 2w+1, 2w+2\}$, one can simply try every integer between 2 and $2w+2$ and verify whether it meets one of the above requirements. Assuming that division can be done in constant time, this procedure takes time at most

$O(w)$. As for the construction problem, once a suitable d is found, is it easy to construct a minimal period p satisfying Theorem 2. The main bottleneck here is to construct the output, which can be done in time $O(n)$. □

4.2 Heterogeneous Preference Types

We begin with a straightforward result about the case $w = 1$.

Theorem 4. *For half-window size $w = 1$, given the number of agents of each preference type, a valid configuration, if one exists, can be found in $\Theta(n)$ time.*

Proof. We will develop necessary and sufficient conditions for a valid configuration to exist for a given collection of agents with given preference types. In any configuration of agents, the agents must occur in an even number of *runs*, with each run consisting of some number of agents of the same color: we will refer to them as blue or red runs. If there are k runs, then starting from a fixed point at the boundary of two runs we can number the runs around the circle as S_0, S_1, ..., S_{k-1} with the odd-numbered runs being blue and the even numbered ones being red.

Consider a valid configuration. We note that that a blue run in the configuration consists of a single agent if and only if it is of type 0. Consider a blue run containing $k \geq 2$ agents: the $(k-2)$ agents in the interior of the run must all be of type 2 and the two agents at the run boundary are or type 1. Thus,

$$b_2 > 0 \implies b_1 \geq 2 \text{ and } r_2 > 0 \implies r_1 \geq 2 \tag{1}$$

and since the agents of type 1 occur in pairs of the same color, b_1 and r_1 are even and the following necessary conditions hold:

$$b_0 + \frac{b_1}{2} = \frac{K}{2} = r_0 + \frac{r_1}{2} \tag{2}$$

Together, these conditions are necessary to guarantee a valid configuration. In fact, they are also sufficient: given a collection of agents that satisfies the above constraints, we can construct a configuration of *interleaved* blue and red runs as follows. Starting at a reference point on the circle, the first b_0 blue runs (resp. r_0 red runs) consist of singleton blue agents of type 0 (resp. red agents of type 0). The first non-singleton blue (or red run) is special: it has two blue agents of type 1 (resp. red agents of type 1) at the ends of the run and *all* the blue agents of type 2 (resp. red agents of type 2) in its interior. Now, the remaining blue (or red) runs are each configured to have exactly two blue agents of type 1 (resp. two red agents of type 1). □

Next we consider the case when agents have heterogeneous preference types for arbitrary window sizes. Recall that b_i (resp. r_i) is the number of blue (resp. red) agents that have preference i. The neighborhood of a blue agent with preference i must be a string of length $2w + 1$ with the $(w + 1)^{th}$ symbol being B and with exactly $i + 1$ occurrences of B (that is i occurrences other than the middle symbol). Similarly, the neighborhood of a red agent with preference i is a string of length $2w+1$ with the $(w+1)^{th}$ symbol being R and with exactly $i+1$ occurrences of R.

Given a valid configuration $\mathcal{C}$, that is, one that satisfies all the agents' preferences, let S be the set of all substrings (interpreted circularly as usual) of $\mathcal{C}$ of length $2w+1$. Furthermore, let $P_i \subseteq S$ and $Q_i \subseteq S$ denote the set of strings denoting neighborhoods of blue and red agents (resp.) of preference type i in the given configuration. Finally, for every $y \in S$, let X_y be the number of occurrences of y in $\mathcal{C}$, that is, the number of agents in the configuration whose neighborhood is represented by the string y.

We now present a set of inequalities and equalities concerning the quantities X_y. First, by definition,

$$X_y > 0 \text{ for every } y \in S. \tag{3}$$

Next, observe that since $\mathcal{C}$ is a valid configuration, for every $i = 0, 1, \ldots 2w$ with $b_i \neq 0$,

$$\sum_{y \in P_i} X_y = b_i. \tag{4}$$

Similarly for every $i = 0, 1, \ldots 2w$ with $r_i \neq 0$

$$\sum_{y \in Q_i} X_y = r_i. \tag{5}$$

Let T be the set of all substrings of length $2w$ of $\mathcal{C}$ (substrings interpreted circularly). Equivalently, T is the set of the strings obtained by deleting either the first or the last symbol of a string in S. Consider $z \in T$. Since the string z can be preceded and succeeded by an R or a B, it follows that for each $z \in T$

$$X_{Rz} + X_{Bz} = X_{zR} + X_{zB}. \tag{6}$$

Note that if any of the strings Rz, Bz, zR, zB do not exist in S, we simply remove the corresponding X quantity from the above equality, thereby preserving (3).

We define a De Bruijn multi-digraph G_S with respect to the set S, as described below. For every string z in T, we have a corresponding node in G. For every string $y \in S$, we create X_y directed edges in G_S all labeled y, from the node s to the node t where s and t are the strings obtained by dropping the first and the last symbols of y respectively. Now observe that the out-degree of a node z is $X_{zR} + X_{zB}$ (if either zR or zB is not in S, simply replace the corresponding term by 0), and its in-degree is $X_{Rz} + X_{Bz}$. By Eq. 6, for every node z, its indegree equals its outdegree. In fact, it is easy to see that the configuration $\mathcal{C}$ corresponds to an Euler tour in the graph G_S.

Thus we have shown the following lemma:

Lemma 4. *If $\mathcal{C}$ is a valid configuration, and S is the set of its substrings of length $2w+1$, then Eqs. 3, 4, 5, and 6 hold, and furthermore, $\mathcal{C}$ corresponds to an Euler tour in the associated De Bruijn multigraph.*

We now describe our algorithm to construct a valid configuration, if it exists. We repeat the following steps for every subset S of strings of length $2w+1$, until a valid configuration is found.

Step 1: Fix a subset S of strings of length $2w+1$. Set up and solve the ILP described by Eqs. 3, 4, 5, and 6, with respect to the set S.
Step 2: If the ILP has a feasible solution, construct the De Bruijn multi-digraph G_S, using the values of the variables X_y in the solution of the ILP.
Step 3: If the digraph G_S is connected, find an Euler tour in G_S, and build a configuration by traversing the Euler tour and appending the middle symbol of each arc's label.

To analyze the complexity of our algorithm, we use the following result of Lokshtanov [15] derived from Lenstra [14] with improvements by Kannan [12] and Frank and Tardos [9] (see [15], Theorem 2.8.2):

Theorem 5 [15]**.** *A solution to an ILP may be found in time $O(p^{2.5p}L\log(N))$ where p is the number of variables in the ILP, L is a bound on the number of bits required to describe the ILP and N is the maximum of the absolute values any variable can take.*

We are ready to prove the main result of this section:

Theorem 6. *For agents with heterogeneous preference types, a valid configuration can be found in $O(n + 2^{2^{2w+1}} w(w2^{2w+1} + \log n)2^{(2w+1)2.5\cdot 2^{2w+1}} \log n)$ time, if one exists. Therefore the problem is FPT for parameter w.*

Proof. The correctness of the algorithm above can be seen as follows. Suppose for a particular subset S of strings, the corresponding ILP has a feasible solution, and the multi-digraph G_S is connected. Since Eq. 6 are satisfied, for every node in the graph G_S, its indegree equals its outdegree. Therefore H_S admits an Euler tour; clearly the Euler tour corresponds to a valid configuration.

If for every set S, either the graph G_S is not connected, or there is no feasible solution to the ILP, then it follows from Lemma 4 that there is no feasible configuration. To see this, it is enough to observe that if $\mathcal{C}$ is a feasible configuration, then for S the set of all substrings of $\mathcal{C}$ of length $2w+1$, G_S is connected and the number of occurrences of each $y \in S$ in the configuration $\mathcal{C}$ form a feasible solution to the ILP.

To analyze the running time, observe that the main cost is in solving the ILP generated for each of the possible subsets of strings of length $2w+1$, i.e., solving $2^{2^{2w+1}}$ ILPs. The ILPs have at most 2^{2w+1} variables. They contain 2^{2w+1} equations of type 3 each requiring $O(w)$ bits to describe. There are $2(2w+1)$ equations of type 4 and 5 each requiring $O(w2^{2w+1} + \log n)$ bits and there are 2^{2w} equations of type 6 each requiring $O(w)$ bits. The total length of an ILP is then $O(w(w2^{2w+1} + \log n))$ and thus by Theorem 5, it can be decided in $O(w(w2^{2w+1} + \log n)2^{(2w+1)2.5\cdot 2^{2w+1}} \log n)$. Constructing the graph and deciding if it is connected takes only an additional $O(2^{2w})$ time so that the overall runtime is $O(2^{2^{2w+1}} w(w2^{2w+1} + \log n)2^{(2w+1)2.5\cdot 2^{2w+1}} \log n)$. □

5 Preference Thresholds

Recall that β_i and ρ_i are the number of blue and red agents respectively with threshold i. We start with a simple result about the case when all agents have the same threshold.

Theorem 7. *Suppose all nodes have the same threshold t. Then there exists a valid configuration if and only if one of the following is true:*

1. $r = 0$ *or* $b = 0$.
2. $t \leq w$ *and* $r, b \geq t + 1$

Therefore, a valid configuration, if one exists, can be found in $O(n)$ time.

Proof. If $r = 0$ (resp. $b = 0$), then clearly any permutation of blue (resp. red) agents satisfies all their preferences. Therefore we assume $r, b > 0$ (there are both red and blue agents). Since each agent requires t or more neighbors of its own color, it is necessary that $r, b \geq t + 1$. Suppose $t > w$ and there is a valid configuration. There must be at least one run of each color. Consider a pair of adjacent agents A and B such that A is red and B is blue. Since A has at least t other red neighbors, at most one of which is not a neighbor of B, and A is an additional red neighbor of B, it follows that B has at least t red neighbors. But then B can have at most $2w - t < w < t$ blue neighbors, a contradiction. We conclude that $t \leq w$. It is easy to see that a single run each of red and blue agents provides a valid configuration. □

Next we consider the special case of heterogeneous thresholds with $w = 1$. This means only three different values of thresholds are possible: 0, 1, 2.

Theorem 8. *For $w = 1$, and heterogeneous preference threshold specifications, a valid configuration can be constructed, if one exists, in $\Theta(n)$ time.*

Proof. If either $b = 0$ or $r = 0$, clearly any permutation of all blue agents satisfies all their preferences. So assume $r, b > 0$. Then we must have at least one run of each color. The endpoints of each run must be agents of threshold either 0 or 1. That is, we need:

$$\rho_0 + \rho_1 \geq 2 \text{ and } \beta_0 + \beta_1 \geq 2$$

If this condition is satisfied, the following configuration is valid: create one run of each color, the endpoints of the runs are agents with threshold 0 or 1, the other agents are placed inside the runs in an arbitrary fashion. □

Finally, we consider the general case when agents have heterogeneous preference thresholds, for arbitrary values of w. Any blue agent with preference threshold $B_{\geq i}$ (resp. any red agent with preference threshold $R_{\geq i}$), for some $i : \ 0 \leq i \leq 2w,$ requires *at least* i blue (resp. red) agents in its neighborhood. Any such blue (resp. red) agent has its neighborhood constrained to be a string in P_k (respectively Q_k) where $i \leq k \leq 2w$ (recall that P_k and Q_k respectively denote all possible neighborhoods of blue and red agents containing exactly k

proper neighbors of their own color in a given configuration). This suggests modifying the ILP above by replacing the equalities in (4) with inequalities, one for each $0 \leq i \leq 2w$:

$$\sum_{y \in \cup_{i \leq k \leq 2w} P_k} X_y \geq \sum_{i \leq k \leq 2w} \beta_k \tag{4-i}$$

and replacing (5) likewise with:

$$\sum_{y \in \cup_{i \leq k \leq 2w} Q_k} X_y \geq \sum_{i \leq k \leq 2w} \rho_k \tag{5-i}$$

with Eqs. (4-i) and (5-i) above being satisfied *with equality*. It is straightforward to see that the complexity of solving the problem is exactly the same as for heterogeneous preference types, as it involves solving a different ILP with the same number of variables and constraints. The theorem below follows:

Theorem 9. *For agents with heterogeneous threshold types, a valid configuration can be found in* $O(n + 2^{2^{2w+1}} w(w2^{2w+1} + \log n)2^{(2w+1)2.5 \cdot 2^{2w+1}} \log n)$ *time, if one exists. Therefore it is FPT for parameter* w.

6 Discussion

We considered three ways of specifying seating preferences and for each of these three kinds of specifications, we gave algorithms to construct such an arrangement when possible. Our main result is that satisfying seating preferences is fixed-parameter tractable (FPT) with respect to half-window size parameter w for preference types and thresholds, while it can be solved in linear time for preference lists. We also gave linear time algorithms for some special cases. We remark that if the input is given simply as a set of distinct preference specifications, and the number of agents that desire them, the existence of a valid configuration can be determined in $O(w)$ time for homogeneous preference types and in $O(2^{2^{2w+1}} w(w2^{2w+1} + \log n)2^{(2w+1)2.5 \cdot 2^{2w+1}} \log n)$ time for heterogeneous preference types and thresholds.

The existence of a polynomial algorithm in both w and n for heterogeneous preference types and thresholds remains open, as does the construction of valid configurations for grids. It would be interesting to solve the general case when each agent independently decides the manner of expressing its preference. Another direction of interest would be to start with a given configuration, and *move* the agents in an efficient manner to final positions satisfying their preferences.

References

1. Andrews, J.A., Jacobson, M.S.: On a generalization of chromatic number. Congr. Numer. **47**, 33–48 (1985)
2. Benard, S., Willer, R.: A wealth and status-based model of residential segregation. Math. Sociol. **31**(2), 149–174 (2007)

3. Benenson, I., Hatna, E., Or, E.: From schelling to spatially explicit modeling of urban ethnic and economic residential dynamics. Sociol. Methods Res. **37**(4), 463–497 (2009)
4. Brandt, C., Immorlica, N., Kamath, G., Kleinberg, R.: An analysis of one-dimensional Schelling segregation. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, pp. 789–804. ACM (2012)
5. Cowen, L.J., Cowen, R.H., Woodall, D.R.: Defective colorings of graphs in surfaces: partitions into subgraphs of bounded valency. J. Gr. Theory **10**(2), 187–195 (1986)
6. Cowen, L.J., Goddard, W., Jesurum, C.E.: Coloring with defect. In: SODA, pp. 548–557 (1997)
7. Cowen, L.J., Goddard, W., Jesurum, C.E.: Defective coloring revisited. J. Gr. Theory **24**(3), 205–219 (1997)
8. Dall'Asta, L., Castellano, C., Marsili, M.: Statistical physics of the schelling model of segregation. J. Stat. Mech: Theory Exp. **2008**(07), L07002 (2008)
9. Frank, A., Tardos, É.: An application of simultaneous diophantine approximation in combinatorial optimization. Combinatorica **7**(1), 49–65 (1987)
10. Henry, A.D., Pralat, P., Zhang, C.-Q.: Emergence of segregation in evolving social networks. Proc. Natl. Acad. Sci. **108**(21), 8605–8610 (2011)
11. Immorlica, N., Kleinberg, R., Lucier, B., Zadomighaddam, M.: Exponential segregation in a two-dimensional schelling model with tolerant individuals. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 984–993. SIAM (2017)
12. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)
13. Kuhn, F.: Weak graph colorings: distributed algorithms and applications. In: Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, pp. 138–144. ACM (2009)
14. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
15. Lokshtanov, D.: New methods in parameterized algorithms and complexity. Ph.D. thesis. University of Bergen, Norway (2009)
16. Pancs, R., Vriend, N.J.: Schelling's spatial proximity model of segregation revisited. J. Public Econ. **91**(1), 1–24 (2007)
17. Parshina, O.G.: Perfect 2-colorings of infinite circulant graphs with continuous set of distances. J. Appl. Ind. Math. **8**(3), 357–361 (2014)
18. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. **98**(17), 9748–9753 (2001)
19. Schelling, T.C.: Models of segregation. Am. Econ. Rev. **59**(2), 488–493 (1969)
20. Schelling, T.C.: Dynamic models of segregation. J. Math. Sociol. **1**(2), 143–186 (1971)
21. Young, H.P.: Individual Strategy and Social Structure: An Evolutionary Theory of Institutions. Princeton University Press, Princeton (2001)
22. Zhang, J.: A dynamic model of residential segregation. J. Math. Sociol. **28**(3), 147–170 (2004)

Two-Dimensional Knapsack for Circles

Carla Negri Lintzmayer[1](✉), Flávio Keidi Miyazawa[2], and Eduardo Candido Xavier[2]

[1] Center for Mathematics, Computer, and Cognition, Federal University of ABC, Santo André, Brazil
`carla.negri@ufabc.edu.br`

[2] Institute of Computing, University of Campinas, Campinas, Brazil
`{fkm,eduardo}@ic.unicamp.br`

Abstract. In this paper we consider the Two-dimensional Knapsack for Circles problem, in which we are given a set $\mathcal{C}$ of circles and want to pack a subset $\mathcal{C}' \subseteq \mathcal{C}$ of them into a rectangular bin of dimensions w and h such that the sum of the area of circles in $\mathcal{C}'$ is maximum. By packing we mean that the circles do not overlap and they are fully contained inside the bin. We present a polynomial-time approximation scheme that, for any $\epsilon > 0$, gives an approximation algorithm that packs a subset of the input circles into an augmented bin of dimensions w and $(1 + O(\epsilon))h$ such that the area packed is at least $(1 - O(\epsilon))$ times the area packed by an optimal solution into the regular bin of dimensions w and h. This result also extends to the multiple knapsack version of this problem.

Keywords: Circle packing · Two-dimensional Knapsack
Polynomial-time approximation scheme

1 Introduction

In the *Two-dimensional Knapsack* problem, we are given a set of 2D items with profits and we want to pack some of the items into one 2D recipient such that the sum of their profits is maximum. By packing we mean that the items do not overlap and they are fully contained inside the recipient. We consider the variation where the items are circles, the item profit is the circle's area, and the recipient is a rectangle.

The Two-dimensional Knapsack is a generalization of the famous *Knapsack* problem in one dimension, where we have a set of items with profits and weights and want to find a subset of items of maximum profit whose total weight is at most the capacity of the given knapsack. As such, it is also NP-hard [6]. This problem is also related to the likewise famous *Two-dimensional Bin Packing*

This work was supported by São Paulo Research Foundation (grants 2016/14132-4, 2015/11937-9, 2016/23552-7, 2016/01860-1) and National Counsel of Technological and Scientific Development (grants 306358/2014-0, 311499/2014-7, and 425340/2016-3).

problem, in which we are given a set of 2D items and we want to pack them all into the minimum number of given 2D bins.

Problems in which we have to pack circles have applications in crystallography, error-correcting codes, coverage of a geographical area with cell transmitters, storage of cylindrical barrels, packaging bottles or cans [15], and origami design [3]. Unfortunately, Demaine *et al.* [3] showed that it is NP-hard to decide whether a given set of circles can be packed into a rectangle, an equilateral triangle, or a unit square.

There are a number of results in the literature for the Two-dimensional Bin Packing [2,12]. When the items are circles, several results involve packing circles of same radii into squares [15], but when the circles have different radii there are not that many [7]. Miyazawa *et al.* [14] showed an asymptotic PTAS for packing circles into rectangles when one can augment one direction of the bin by a small constant. Hokama *et al.* [8] gave a 2.439-competitive algorithm for the online circle packing into squares, in which one circle is given at a time.

Regarding Two-dimensional Knapsack problems, there are also several results in the literature, but, to the best of our knowledge, none of them considers that the items are circles. Fishkin *et al.* [4] showed a PTAS for packing squares into a unit square bin that is augmented in both dimensions by a factor of ϵ. The same authors [5] showed later two PTASs for packing rectangles into a unit square bin. One of them considers the area of the rectangles as profits and the other considers general profits, but the bin is augmented in both dimensions by a factor of ϵ. Then Jansen and Solis-Oba [9] showed a PTAS for packing squares into a rectangular bin. For packing rectangles with profits the best-known results are a $1/(2+\epsilon)$-approximation algorithm due to Jansen and Zhang [10] and a $1/(1+\epsilon)$-approximation algorithm that runs in quasi-polynomial time due to Adamaszek and Wiese [1].

Our contributions. We consider the *Two-dimensional Knapsack for Circles (2DK)* when items profits are equal to their areas. We use the structure of solutions presented by Miyazawa *et al.* [14] combined with new ideas to obtain a PTAS for this problem such that, for any given $0 < \epsilon \leq 1$, there is a $(1 - O(\epsilon))$-approximation algorithm that packs a subset of circles of the input into a bin of augmented height (an extra factor of $O(\epsilon)$).

The rest of the paper is divided as follows. Section 2 presents the main definitions that we will need. Section 3 gives an overview of an algorithm for 2DK and how the analysis will be performed. Such algorithm, which has exponential-time, is presented in Sect. 4. Section 5 shows how to transform an optimal solution for 2DK so that it has the same structure of the one built by the algorithm. The profit of such transformed optimal solution is then related to the profit of the algorithm's solution in Sect. 6. Section 7 shows an algorithm for a generalized problem, which is needed to improve the time complexity of the previous algorithm for 2DK. We finally show in Sect. 8 that such algorithm is a PTAS for 2DK.

2 Definitions

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ be a set of n circles. The radius of $c_i \in \mathcal{C}$ is denoted as r_i, with $r_i \in \mathbb{Q}_+$, while its *profit*, denoted as $p(c_i)$, is defined as its area πr_i^2. For any object X (a set of circles, a bin, or a set of bins), we denote as $p(X)$ the sum of the profit of circles contained in X. We denote a bin B (knapsack) of width $w \in \mathbb{Q}_+$ and height $h \in \mathbb{Q}_+$ as $B_{w\times h}$ and say it *has size* $w \times h$. We may omit the dimensions from the notation when they are clear from the context.

A *packing* of a set $\mathcal{C}$ of circles into a bin $B_{w\times h}$ is a sequence of pairs $(x_i, y_i) \in \mathbb{R}_+^2$, one for each $c_i \in \mathcal{C}$, which describe the center coordinates of the circles and satisfy the constraints: (1) two circles cannot overlap, that is, for any $c_i, c_j \in \mathcal{C}$, with $i \neq j$, $(x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2$, and (2) all circles must be fully inside the bin, that is, for all $c_i \in \mathcal{C}$, $r_i \leq x_i \leq w - r_i$ and $r_i \leq y_i \leq h - r_i$.

An instance for the *Two-dimensional Knapsack for Circles (2DK)* is a triple $(\mathcal{C}, w, h)$, where $\mathcal{C}$ is a set of n circles, $w \leq h$, and for each $c_i \in \mathcal{C}$, $2r_i \leq w$. The objective of this problem is to pack a subset $\mathcal{C}' \subseteq \mathcal{C}$ of circles into a bin $B_{w\times h}$ such that the profit $p(\mathcal{C}')$ is maximum. The profit of an optimal solution for the 2DK, given an input $(\mathcal{C}, w, h)$, is denoted by $\mathrm{OPT}_{w\times h}(\mathcal{C})$. For the rest of the paper, we consider that $p(\mathcal{C}) > \pi wh/8$ in any instance of 2DK. Lemma 1 shows what to do when $p(\mathcal{C}) \leq \pi wh/8$.

Lemma 1. *Let $(\mathcal{C}, w, h)$ be an instance for 2DK. If $p(\mathcal{C}) \leq \pi wh/8$, all circles of $\mathcal{C}$ can be packed into one bin $B_{w\times h}$.*

Proof. Place each $c_i \in \mathcal{C}^*$ into a square of side $2r_i$. If the total area of such squares is at most $wh/2$, they can all be packed into $B_{w\times h}$, according to Meir and Moser [13]. Since $\pi/4$ of the area of each square is filled with a circle, we have that if $p(\mathcal{C}) \leq \frac{\pi}{4} \times \frac{wh}{2} = \frac{\pi wh}{8}$, then all circles can be packed into the bin. $\square$

We also consider a generalized problem, called *Two-dimensional Multiple Knapsack for Circles (2DMK)*, whose instances are quadruples $(\mathcal{C}, w, h, f)$, where $\mathcal{C}$, w, and h are as in 2DK, and $f \in \mathbb{Z}_+$. The objective is to pack $\mathcal{C}' \subseteq \mathcal{C}$ into f bins of size $w \times h$ such that $p(\mathcal{C}')$ is maximum. The profit of an optimal solution for the 2DMK, given an input $(\mathcal{C}, w, h, f)$, is denoted by $\mathrm{OPT}^{\mathrm{2DMK}}_{w\times h}(\mathcal{C}, f)$.

3 Overview of the Algorithm for 2DK and its Analysis

Our algorithm for 2DK will receive as input a tuple $(\mathcal{C}, w, h, \varepsilon)$, where $(\mathcal{C}, w, h)$ is an input for 2DK, $\varepsilon \in \mathbb{R}_+$ is a constant such that $0 < \varepsilon < 1$, $r = 1/\varepsilon$ is integer multiple of 2, and $h/(w\varepsilon)$ is integer. From now on, $\mathcal{C}$, w, h, and ε always have such meanings. Also, all other notations defined in this section will be used along the text.

We first need to partition $\mathcal{C}$ into a sequence of sets such that the ith set has circles with radii much larger than the circles of the $(i+1)$th set. Formally, let $\mathcal{C}^* \subseteq \mathcal{C}$ be the set of circles of an optimal solution for input $(\mathcal{C}, w, h)$ of 2DK. For every integer $i \geq 0$, let $\mathcal{G}_i = \{c_j \in \mathcal{C} : \varepsilon^{2i} w \geq 2r_j > \varepsilon^{2(i+1)} w\}$ and $\mathcal{G}_i^* = \mathcal{G}_i \cap \mathcal{C}^*$.

Now for each integer j such that $0 \leq j < r$, let $\mathcal{H}_j = \{c_\ell \in \mathcal{G}_i : i \equiv j \pmod{r}\}$ and $\mathcal{H}^*_j = \mathcal{H}_j \cap \mathcal{C}^*$. Let t be an integer such that $p(\mathcal{H}^*_t) \leq \varepsilon p(\mathcal{C}^*)$. Note that this is possible because there are $r = 1/\varepsilon$ such sets. Now remove $\mathcal{H}_t$ from $\mathcal{C}$ and $\mathcal{H}^*_t$ from $\mathcal{C}^*$. For every integer $j \geq 0$, let $\mathcal{S}_j = \bigcup_{i=t+(j-1)r+1}^{t+jr-1} \mathcal{G}_i$ and $\mathcal{S}^*_j = \mathcal{S}_j \cap \mathcal{C}^*$. Note that the minimum diameter of a circle in $\mathcal{S}_j$ is $\varepsilon^{2(t+jr)}w$ while the maximum diameter of a circle in $\mathcal{S}_{j+1}$ is $\varepsilon^{2(t+jr+1)}w$, so their sizes are different by a factor of ε^2. Thus, $\mathcal{S}_0, \mathcal{S}_1, \ldots$ is the partition we needed. Because of this, let $w_0 = w$, $h_0 = h$, and $w_j = h_j = \varepsilon^{2(t+(j-1)r)+1}w$ for every $j \geq 1$[1]. This means that there is a gap between the sets: any circle in $\mathcal{S}_j$ is at most a factor of ε smaller than a (squared) bin of size $w_j \times h_j$, and such bin is also smaller than any circle of the set $\mathcal{S}_{j-1}$ by a factor of ε.

The main idea of our algorithm (given Sect. 4) is to pack circles of set $\mathcal{S}_j$ into bins of size $w_j \times h_j$. In the first iteration, circles of $\mathcal{S}_0$ are packed into one $B_{w\times h}$, as $w = w_0$ and $h = h_0$. The remaining space not used by these circles will be used to pack smaller circles from other sets $\mathcal{S}_j$, starting with $j = 1$ and considering one set per iteration. For that, we create a grid over the bins that packed circles from $\mathcal{S}_{j-1}$ that divides them into subbins of size $w_j \times h_j$ and we check which subbins do not intersect previously packed circles of $\mathcal{S}_{j-1}$ so that they can be used to pack circles of $\mathcal{S}_j$. Note that $w_j = w_{j+1}/\varepsilon^{2r}$, and since both $1/\varepsilon$ and $h/(w\varepsilon)$ are integers, a bin of size $w_j \times h_j$ can be perfectly divided into bins of size $w_{j+1} \times h_{j+1}$, for any $j \geq 0$. We will say "grid of size $a \times b$" when we mean "grid that divides the bin into subbins of size $a \times b$". Also, we denote by $\mathsf{G}_j(B)$ the subbins in the grid of size $w_j \times h_j$ over some bin B or set B of bins.

Thus, the solution has a well-behaved structure: if we consider a grid of size $w_j \times h_j$, for all $j \geq 1$, over the input bin $B_{w\times h}$, then the circles of set $\mathcal{S}_j$ that belong to the solution are fully contained inside elements of such grid. In our analysis, we first show how to transform a given optimal solution by discarding some circles such that, in the end, all circles of set $\mathcal{S}^*_j$ that belong to the transformed solution are fully contained inside elements of a grid of size $w_j \times h_j$ over the input bin (Sect. 5). Then we show that in the end of this transformation, the total area of discarded circles is at most $O(\varepsilon)$. The final part of the analysis shows that the profit of the solution created by our algorithm is not so far from the profit of the transformed optimal solution (Sect. 6).

4 Algorithm for Two-Dimensional Knapsack for Circles

We call our algorithm CircleKnapsack and its pseudocode is given below, followed by an explanation of its behavior.

```
1: function CircleKnapsack($\mathcal{C}$, $w$, $h$, $\varepsilon$)
2:     Create sets $\mathcal{G}_i$, $i \geq 0$, and $\mathcal{H}_j$, $0 \leq j < r$, as described in Section 3
3:     for each $t$ between 0 and $r - 1$ such that $p(\mathcal{H}_t) \leq \varepsilon p(\mathcal{C})$ do
4:         Create sets $\mathcal{S}_j$, $j \geq 0$, as described in Section 3
```

[1] Note that $w_j = h_j$ for all $j \geq 1$, but w_0 is not necessarily equal to h_0. This is the only reason we keep using w_j and h_j throughout the rest of the text.

```
5:         $F_0^t \leftarrow \{B_{w\times h}\}$
6:         for all $j \geq 0$ do
7:             $F_j^t \leftarrow$ an optimal solution for 2DMK with input $(\mathcal{S}_j, w_j, h_j, |F_j^t|)$
8:             $F_{j+1}^t \leftarrow$ bins of $\mathsf{G}_{j+1}(F_j^t)$ that do not intersect circles of $\mathcal{S}_j$
9:     Choose $t$ such that $p(\bigcup_{j\geq 0} F_j^t)$ is maximum
10:    return Packing of $F_0^t, F_1^t, \ldots$ into $B_{w\times h}$
```

CircleKnapsack receives the input $(\mathcal{C}, w, h, \varepsilon)$. In Sect. 3, we removed set $\mathcal{H}_t$ such that $p(\mathcal{H}_t^*) \leq \varepsilon p(\mathcal{C}^*)$, which is necessary to create sets $\mathcal{S}_j$, for $j \geq 0$. Since we do not know the value of an optimal solution, in line 3 we try all possible values for t, which eventually reach the one related to the optimal solution (because $\varepsilon p(\mathcal{C}^*) \leq \varepsilon p(\mathcal{C})$). In line 10, therefore, we return the best solution found (created between lines 4 and 8) among all iterations. We thus describe now how our algorithm generates one solution given a fixed value of t.

For each integer $j \geq 0$, our algorithm keeps a set F_j^t of free bins of size $w_j \times h_j$, which are used to pack circles of set $\mathcal{S}_j$. These are subbins of the initial bin. When $j = 0$ the only free bin is $B_{w\times h}$ (line 5). In line 7 we choose the circles of $\mathcal{S}_j$ that will be packed into the $|F_j^t|$ available bins. Our wish is to do a greedy choice, which is to pack in the bins of size $w_j \times h_j$ of F_j^t a subset of circles of $\mathcal{S}_j$ such that $p(F_j^t)$ is maximum. Note that this is the definition of the generalized problem 2DMK, and so we use an optimal solution for the same.

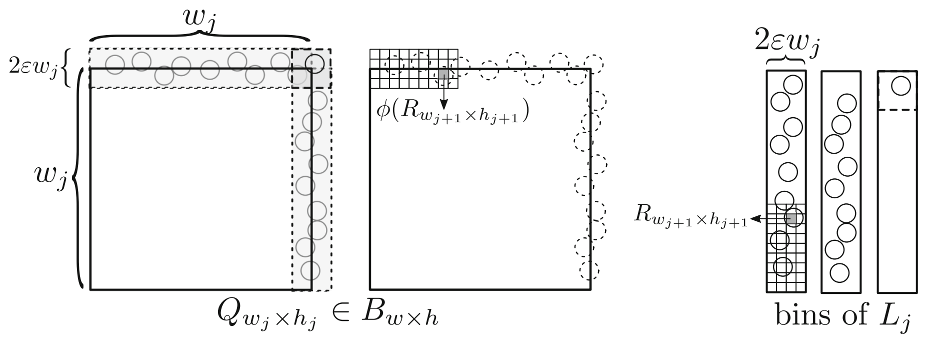
It remains to find some space to pack circles of the next set, $\mathcal{S}_{j+1}$. This is done in line 8, in which we create a grid of size $w_{j+1} \times h_{j+1}$ over each of the current bins of set F_j^t. Any element of this grid that does not intersect a packed circle of $\mathcal{S}_j$ is then placed into set F_{j+1}^t so it can be used in the next iteration.

5 Modifying an Optimal Solution

Consider input $(\mathcal{C}, w, h)$ for 2DK, let $\mathcal{C}^* \subseteq \mathcal{C}$ be a set of circles of an optimal solution, i.e., $p(\mathcal{C}^*) = \mathrm{OPT}_{w\times h}(\mathcal{C})$, and let $B_{w\times h}^*$ be the bin where $\mathcal{C}^*$ is packed. Recall that in Sect. 3 we defined sets $\mathcal{G}_i^*$ for $i \geq 0$, $\mathcal{H}_j^*$ for $0 \leq j < r$, and, after finding and removing $\mathcal{H}_t^*$ such that $p(\mathcal{H}_t^*) \leq \varepsilon p(\mathcal{C}^*)$, we defined sets $\mathcal{S}_j^*$ for $j \geq 0$. In this section we describe how to transform a packing of $\mathcal{C}^*$ into $B_{w\times h}^*$ into a structured packing where circles of set $\mathcal{S}_j^*$, for $j \geq 0$, are fully contained inside bins of $\mathsf{G}_j(B_{w\times h}^*)$. This algorithm is called TransformOptimal. It has two phases, as we will see next, and it maintains a set $\mathcal{C}^\star$, initially equal to $\mathcal{C}^*$, which keeps the circles that remain in the transformed solution.

We say that *grid j over $B_{w_k\times h_k}$* is a grid of size $w_j \times h_j$ over bin $B_{w_k\times h_k}$. The *lines* of grid j over $B_{w_k\times h_k}$ are the w_k/w_j vertical lines and h_k/h_j horizontal lines that define the subbins of the grid. For any bin Y (or set Y of bins), we will denote as $\mathsf{Area}(Y)$ the area of Y (or the sum of the areas of the bins in Y).

The first phase of the algorithm deals with the main problem of the initial packing of $\mathcal{C}^*$, which is the fact that for any grid j over $B_{w\times h}^*$, circles of $\mathcal{S}_j^*$ may intersect the grid lines. Note that we only have to consider $j \geq 1$, because in the initial packing circles of $\mathcal{S}_0^*$ are already fully contained into the bin $B_{w\times h}^*$.



w_j
$2\varepsilon w_j$
w_j
$\phi(R_{w_{j+1}\times h_{j+1}})$
$Q_{w_j\times h_j} \in B_{w\times h}$
$2\varepsilon w_j$
$R_{w_{j+1}\times h_{j+1}}$
bins of L_j

Fig. 1. Circles from $\mathcal{W}_j$ that intersect the borders of bin $Q_{w_j\times h_j}$ are removed from their current positions and placed into three temporary bins of size $2\varepsilon w_j \times h_j$.

Let $\mathcal{W}_j$ be the set of circles of $\mathcal{S}_j^*$ that intersect the lines of grid j over $B^*_{w\times h}$. In the end of the first phase, each circle of $\mathcal{W}_j$ will either be reallocated inside $B^*_{w\times h}$ or discarded. We start by temporarily removing such circles from $B^*_{w\times h}$ and placing them into a set L_j of bins in the following way. For each subbin $Q_{w_j\times h_j}$ of $\mathsf{G}_j(B^*_{w\times h})$, we only need to care for removing circles that intersect its superior and right borders (doing this for all subbins of the grid will collectively free the grid lines). See Fig. 1 for a visual depiction of the following discussion.

Since all circles of $\mathcal{W}_j$ have diameter at most εw_j, two bins of size $2\varepsilon w_j \times h_j$ comprise almost all circles that intersect the two mentioned borders of $Q_{w_j\times h_j}$. A third bin of size $2\varepsilon w_j \times 2\varepsilon w_j$ is needed to comprise circles from the top right corner. Thus, all circles from these borders can fit into three bins of size $2\varepsilon w_j \times h_j$ each, so we remove them from the current packing, place them into these bins preserving the relative positions, and add all these bins to L_j. Lemma 2 shows the total area of the bins in L_j.

Lemma 2. *For each $j \geq 1$, $\mathsf{Area}(L_j) = 6\varepsilon wh$.*

Proof. There are $\frac{w}{w_j}\frac{h}{h_j}$ subbins in $\mathsf{G}_j(B_{w\times h})$ and each of them contributes to L_j with three rectangles of area $2\varepsilon w_j h_j$ each. □

Now we create a new bin D of size $w \times 6\varepsilon h$. Note that since both w and $6\varepsilon h$ are multiple of w_j, any grid j over D perfectly divides D into bins of size $w_j \times h_j$. Also note that every $1/(2\varepsilon)$ bins of L_j form a bin of size $w_j \times h_j$. Thus, by Lemma 2, all bins of size $w_j \times h_j$ formed from L_j can be packed into D. The idea now for each of these bins is to (i) move it to D, (ii) move it back to $B^*_{w\times h}$, or (iii) discard it.

Suppose we pack all bins of size $w_j \times h_j$ formed from L_j into D and consider grid $j+1$ over $B^*_{w\times h}$ and over D. Since $2\varepsilon w_j$ is a multiple of $w_{j+1} = \varepsilon^{2r} w_j$, we can make a bijection ϕ between the subbins of $\mathsf{G}_{j+1}(D)$ and the subbins of $\mathsf{G}_{j+1}(B^*_{w\times h})$ that are in the rectangles of size $w_j \times 2\varepsilon w_j$ around the superior and right borders of each $Q_{w_j\times h_j} \in \mathsf{G}_j(B^*_{w\times h})$ (the same ones considered to build L_j). So, for all subbin $R_{w_{j+1}\times h_{j+1}}$ in D there exists a corresponding subbin

$\phi(R_{w_{j+1}\times h_{j+1}})$ in $B^*_{w\times h}$. See Fig. 1. More importantly, if $R_{w_{j+1}\times h_{j+1}}$ is a subbin of D fully contained in a circle of $\mathcal{W}_j$, then $\phi(R_{w_{j+1}\times h_{j+1}})$ is empty in $B^*_{w\times h}$.

During the process described next, consider the following notation. Suppose we pack all bins from L_j into D. A subbin $R \in \mathsf{G}_j(D)$ and the circles packed in it are *real* if they are indeed packed in D, they are *ϕ-virtual* if they were moved to $\phi(R)$, or they are *virtual* if they were removed from D.

The process starts by moving each bin of L_1 to D, which is initially empty. These bins and $\mathcal{W}_1$ are real. The following steps are repeated for each $j \geq 2$. Pack all bins of L_j into D, possibly generating overlaps of circles. For each bin $R = R_{w_j\times h_j} \in \mathsf{G}_j(D)$, we have three possibilities. If R is completely inside a (virtual or real) circle of $\mathcal{W}_{j-1}$, then move the packing in R to $\phi(R)$ and keep it as ϕ-virtual in D. If R intersects the borders of (virtual or real) circles of $\mathcal{W}_{j-1}$, then remove it and keep it as virtual in D. Otherwise, R does not intersect circles of $\mathcal{W}_{j-1}$ and so we have three subcases; let $S = S_{w_{j-1}\times h_{j-1}} \in \mathsf{G}_{j-1}(D)$ be the bin that contains R: (i) if S is real in D, then R can be kept in D because it does not overlap any circle, (ii) if S is ϕ-virtual, then move the packing in R to $\phi(R)$, which does not intersect any circle in $\phi(S)$, and keep R as ϕ-virtual in D, (iii) otherwise S is virtual, so remove R and keep it as virtual (note that the removal of R was already accounted in the removal of S).

At the end of the first phase of the algorithm, we simply discard all circles that are contained in D from $\mathcal{C}^\star$. At this point, we have that the remaining circles are in such way that for every $j \geq 0$, $\mathcal{S}^*_j$ is packed into a subset of $\mathsf{G}_j(B^*_{w\times h})$. We can still have the case, though, that bins from a grid j' over $B^*_{w\times h}$ intersect larger circles, from an $\mathcal{S}^*_j$ with $j' \geq j$. Phase two of the algorithm deals with this problem[2]. It just removes from $\mathcal{C}^\star$ the circles contained into bins of the grid j over $B^*_{w\times h}$ that intersect the borders of circles of $\mathcal{S}^*_{j-1}$, for all $j \geq 1$.

The following is the pseudocode for TransformOptimal.

```
1: function TransformOptimal($\mathcal{C}^*$, $w$, $h$, $\varepsilon$)
2:     $\mathcal{C}^\star \leftarrow \mathcal{C}^*$
3:     Create sets $\mathcal{G}^*_i$, $i \geq 0$, and $\mathcal{H}^*_j$, $0 \leq j < r$, as described in Section 3
4:     Find integer $t$ such that $p(\mathcal{H}^*_t) \leq \varepsilon\mathrm{OPT}_{w\times h}(\mathcal{C})$
5:     Remove all circles of $\mathcal{H}^*_t$ from $\mathcal{C}^\star$
6:     Create sets $\mathcal{S}^*_j$, $j \geq 0$, as described in Section 3
7:     Let $D$ be a bin of size $w \times 6\varepsilon h$
8:     for all $j \geq 1$ do
9:         $L_j \leftarrow \emptyset$
10:        for each $Q_{w_j\times h_j} \in \mathsf{G}_j(B^*_{w\times h})$ do
11:            Let $\mathcal{W}_j \subseteq \mathcal{S}^*_j$ be the set of circles that intersect the superior and
           right borders of $Q_{w_j\times h_j}$
12:            Create three bins $T_1$, $T_2$, and $T_3$ of size $2\varepsilon w_j \times h_j$ and place the
           circles of $\mathcal{W}_j$ in them, preserving positions and removing them from $B^*_{w\times h}$
13:            For all $R \in \mathsf{G}_{j+1}(T_i)$, for $1 \leq i \leq 3$, let $\phi(R)$ be the corresponding
           cell in $\mathsf{G}_{j+1}(Q)$
```

[2] Recall that this is a problem because this situation does not happen in a solution created by CircleKnapsack.

```
14:            Add $T_1$, $T_2$, and $T_3$ to $L_j$
15:    Make groups of $1/(2\varepsilon)$ bins of $L_1$ forming new bins of size $w_1 \times h_1$ and put each of them in one subbin of $\mathsf{G}_1(D)$ as real
16:    for all $j \geq 2$ do
17:        Make groups of $1/(2\varepsilon)$ bins of $L_j$ forming new bins of size $w_j \times h_j$ and put each of them in one subbin of $\mathsf{G}_j(D)$
18:        for each $R \in \mathsf{G}_j(D)$ do
19:            if $R$ is completely inside a (virtual/real) circle of $\mathcal{W}_{j-1}$ then
20:                Move the packing in $R$ to $\phi(R)$ and keep it as $\phi$-virtual in $D$
21:            else if $R$ intersects borders of (virtual/real) circles of $\mathcal{W}_{j-1}$ then
22:                Remove $R$ and keep it as virtual in $D$
23:            else
24:                Let $S \in \mathsf{G}_{j-1}(D)$ be the bin that contains $R$
25:                if $S$ is $\phi$-virtual then
26:                    Move the packing in $R$ to $\phi(R)$ and keep it as $\phi$-virtual
27:                else if $S$ is virtual then
28:                    Remove $R$ and keep it as virtual in $D$
29:    Remove all real circles of $D$ from $\mathcal{C}^\star$
30:    for all $j \geq 1$ do
31:        Let $V \subseteq \mathsf{G}_j(B^*_{w\times h})$ be a set of bins that intersect but are not contained in circles of $\mathcal{S}^*_{j-1}$
32:        Remove all circles of $V$ from $\mathcal{C}^\star$
33:    return Structured packing of $\mathcal{C}^\star$ into $B^*_{w\times h}$
```

We can demonstrate by simple inductions that at the end of TransformOptimal, for all $j \geq 0$, no circle in $\mathcal{C}^\star \cap \mathcal{S}^*_j$ intersects lines of grid j over $B^*_{w\times h}$, all circles of $\mathcal{S}^*_j \cap \mathcal{C}^\star$ are packed into subbins of $\mathsf{G}_j(B^*_{w\times h})$, and none of these subbins intersect larger circles from other $\mathcal{S}^*_k$ where $k < j$. Theorem 1 shows that the profit loss due to all removals of circles is at most $O(\varepsilon)\mathrm{OPT}_{w\times h}(\mathcal{C})$. Lemmas 3 and 4 are used by this theorem.

Lemma 3. *When $p(\mathcal{C}) > \pi wh/8$, $\mathrm{OPT}_{w\times h}(\mathcal{C}) \geq \pi wh/16$.*

Proof. Place each $c_i \in \mathcal{C}$ into a square of side $2r_i$. Now use NFDH [13] to pack all these squares into bins of size $w \times h$. NFDH guarantees that the density of this square packing is 1/4 for each bin, except maybe for the last one. We choose one bin of density 1/4 as a feasible solution for 2DK. Since $\pi/4$ of the area of each square is filled with a circle, an area of at least $\pi wh/16$ of such bin is filled with circles. If no bin has density 1/4, then NFDH packed $\mathcal{C}$ into one bin. Thus the total area of all squares is at most $wh/4$, indicating that the area of the circles is at most $\pi wh/16$, which is a contradiction because $p(\mathcal{C}) > \pi wh/8$. □

Lemma 4. (Miyazawa *et al.* [14]). *For $j \geq 0$, let $c_i \in \mathcal{S}_j$ be a circle packed into a bin B and let $R \subseteq \mathsf{G}_{j+1}(B)$ be the subset of bins that intersect the border of c_i. We have $\mathsf{Area}(R) \leq 16\varepsilon p(c_i)$.*

Theorem 1. TRANSFORMOPTIMAL *transforms a packing of* $\mathcal{C}^* \subseteq \mathcal{C}$ *into bin* $B^*_{w\times h}$*, where* $p(\mathcal{C}^*) = \mathrm{OPT}_{w\times h}(\mathcal{C})$*, into a structured packing of* $\mathcal{C}^\star \subseteq \mathcal{C}^*$ *into the same bin such that* $p(\mathcal{C}^\star) \geq (1 - 64\varepsilon)\mathrm{OPT}_{w\times h}(\mathcal{C})$*.*

Proof (Sketch). We account the profit loss when removing circles from $\mathcal{H}^*_t$ (at most $\varepsilon\mathrm{OPT}_{w\times h}(\mathcal{C})$), from D (at most $96\varepsilon/\pi\mathrm{OPT}_{w\times h}(\mathcal{C}) < 31\varepsilon\mathrm{OPT}_{w\times h}(\mathcal{C})$, by Lemma 3), from virtual bins in D (at most $16\varepsilon \sum_{j\geq 1} p(\mathcal{S}^*_{j-1}) \leq 16\varepsilon\mathrm{OPT}_{w\times h}(\mathcal{C})$, by Lemma 4), and from bins of grid j over $B^*_{w\times h}$ that intersect the borders of circles of $\mathcal{S}^*_{j-1}$ for all $j \geq 1$ (at most $16\varepsilon\mathrm{OPT}_{w\times h}(\mathcal{C})$, by Lemma 4). $\square$

6 Analysis of the Algorithm

Recall that during the execution of CIRCLEKNAPSACK we create sets $F^t_j \subseteq \mathsf{G}_j(B_{w\times h})$ of bins, for all $j \geq 0$, which pack a subset of circles of $\mathcal{S}_j$ using an optimal solution for 2DMK. Thus, by construction,

$$p(\textsc{CircleKnapsack}(\mathcal{C}, w, h, \varepsilon)) = \sum_{j\geq 0} \mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, |F^t_j|). \tag{1}$$

On the other hand, TRANSFORMOPTIMAL receives a packing of $\mathcal{C}^* \subseteq \mathcal{C}$ into $B^*_{w\times h}$ such that $p(\mathcal{C}^*) = \mathrm{OPT}_{w\times h}(\mathcal{C})$ and modify it into a structured packing containing circles from $\mathcal{C}^\star \subseteq \mathcal{C}^*$. For every $j \geq 0$, let $P^\star_j$ be the subset of $\mathsf{G}_j(B^*_{w\times h})$ that is used to pack circles from $\mathcal{S}^*_j \cap \mathcal{C}^\star$. Thus, $\sum_{j\geq 0} p(P^\star_j) = p(\mathcal{C}^\star)$. We showed in Theorem 1 that $p(\mathcal{C}^\star) \geq (1 - 64\varepsilon)\mathrm{OPT}_{w\times h}(\mathcal{C})$. We also have

$$\sum_{j\geq 0} p(P^\star_j) \geq (1 - 64\varepsilon)\mathrm{OPT}_{w\times h}(\mathcal{C}). \tag{2}$$

Lemma 5 thus shows a relation, for $j \geq 0$, between $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, |F^t_j|)$ and $p(P^\star_j)$. Theorem 2 at last shows a bound on the total profit of circles packed by CIRCLEKNAPSACK.

Lemma 5. *For any* $j \geq 0$*,*

$$\sum_{k=0}^{j} \mathrm{OPT}^{\mathrm{2DMK}}_{w_k\times h_k}(\mathcal{S}_k, |F^t_k|) \geq (1 - 16\varepsilon) \sum_{k=0}^{j} p(P^\star_k). \tag{3}$$

Proof. We will show, by induction on j, that (3) is valid. When $j = 0$, $P^\star_0$ is a solution for 2DMK for input $(\mathcal{S}_0, w, h, 1)$, so $\mathrm{OPT}^{\mathrm{2DMK}}_{w\times h}(\mathcal{S}_0, 1) \geq p(P^\star_0)$, and the result follows.

Suppose now that $j \geq 1$. If $|F^t_j| \geq |P^\star_j|$, then we have $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, |F^t_j|) \geq \mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, |P^\star_j|) \geq p(P^\star_j)$. By the induction hypothesis, we have (3) valid for $j - 1$ and so the result follows.

Therefore, consider $|F^t_j| < |P^\star_j|$. As simplification, let us call B the bin of size $w \times h$ returned by CIRCLEKNAPSACK and B^* the bin of size $w \times h$ returned by TRANSFORMOPTIMAL.

Note that a bin $R \in \mathsf{G}_j(B)$ is not in F_j^t if one of these cases happen: (i) R is contained in a circle from some set among $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{j-1}$; (ii) R intersects the border of a circle of $\mathcal{S}_{j-1}$; or (iii) R is contained in a bin $T_{w_k \times h_k}$ that intersects the border of a circle from $\mathcal{S}_{k-1}$, for the largest value of k such that $k < j$.

Let $X \subseteq \mathsf{G}_j(B)$ the bins of case (i). Let $Y \subseteq \mathsf{G}_j(B)$ be the bins of cases (ii) and (iii). Thus, X, Y, and F_j^t is a partition of $\mathsf{G}_j(B)$. Let $Z \subseteq P_j^\star$ be such that $|Z| = |F_j^t|$. Clearly, $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j \times h_j}(\mathcal{S}_j, |F_j^t|) = p(F_j^t) \geq p(Z)$. Let $W \subseteq \mathsf{G}_j(B^*)$ be such that $W \cap Z = \varnothing$ and $|W| = |X|$. Clearly, $p(X) \geq p(W)$. At last, let $V = \mathsf{G}_j(B^*) \backslash (Z \cup W)$. Note that $|V| = |Y|$, which means that $p(V) \leq \mathsf{Area}(V) = \mathsf{Area}(Y) \leq 16\varepsilon \sum_{k=0}^{j} p(F_k^t)$, where the last inequality follows from Lemma 4.

By the definitions above, we have $\sum_{k=0}^{j} \mathrm{OPT}^{\mathrm{2DMK}}_{w_k \times h_k}(\mathcal{S}_k, |F_k^t|) = p(X) + p(Y) + p(F_j^t)$, $\sum_{k=0}^{j} p(P_k^\star) = p(Z) + p(W) + p(V)$, and $p(X) + p(F_j^t) + p(V) \geq p(W) + p(Z) + p(V)$.

Putting it all together, we have $(1 + 16\varepsilon) \sum_{k=0}^{j} \mathrm{OPT}^{\mathrm{2DMK}}_{w_k \times h_k}(\mathcal{S}_k, |F_k^t|) \geq p(X) + p(Y) + p(F_j^t) + p(V) \geq p(X) + p(F_j^t) + p(V) \geq p(W) + p(Z) + p(V) = \sum_{k=0}^{j} p(P_k^\star)$. Since $1/(1 + 16\varepsilon) \geq (1 - 16\varepsilon)$ whenever $\varepsilon > 0$, we have $\sum_{k=0}^{j} \mathrm{OPT}^{\mathrm{2DMK}}_{w_k \times h_k}(\mathcal{S}_k, |F_k^t|) \geq (1 - 16\varepsilon) \sum_{k=0}^{j} p(P_k^\star)$, and the result follows. $\square$

Theorem 2. CircleKnapsack *packs circles from* $\mathcal{C}$ *into a bin of size* $w \times h$ *such that their total profit is at least* $(1 - O(\varepsilon))\mathrm{OPT}_{w \times h}(\mathcal{C})$.

Proof (Sketch). This follows from Eqs. (1), (2), and (3) from Lemma 5. $\square$

7 Algorithm for 2DMK

A careful reader noticed that algorithm CircleKnapsack asks for an optimal solution for 2DMK in each of its iterations, which is an NP-hard problem. On the other hand, we only need solutions for instances $(\mathcal{S}_j, w_j, h_j, |F_j^t|)$, for each $j \geq 0$, where the circles of $\mathcal{S}_j$ have restricted radii. In this section we show how to obtain good solutions for these instances of 2DMK in order to provide a polynomial-time algorithm for 2DK.

Following the work of Miyazawa *et al.* [14], we will also need that the center coordinates of a solution are rational numbers. Since we cannot guarantee that every instance has a rational solution, our solution is in fact a packing of a subset of $\mathcal{C}$ into an augmented bin, of size $w \times (1 + 3\varepsilon)h$. This happens because the packing algorithms given by Miyazawa *et al.* [14] actually obtain coordinates that are roots of polynomial equations, which are possibly irrational, and so they approximate these coordinates and slightly increase the height of the bin. Two important results given by Miyazawa *et al.* [14] will be notably used in this section and are summarized in the next two lemmas. First we need the following definition. A *ξ-packing* of a set $\mathcal{C}$ of circles into a bin $B_{w \times h}$ is a sequence of pairs $(x_i, y_i) \in \mathbb{R}_+^2$ for each $c_i \in \mathcal{C}$ which describe the center coordinates of the circles and satisfy the constraints: (1) two circles may overlap by at most ξ, that is, for any $c_i \in \mathcal{C}$ and $c_j \in \mathcal{C}$, with $i \neq j$, $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq r_i + r_j - \xi$, and (2) the circles can surpass the border of the bin by at most ξ, that is, for all $c_i \in \mathcal{C}$, $r_i - \xi \leq x_i \leq w - r_i + \xi$ and $r_i - \xi \leq y_i \leq h - r_i + \xi$.

Lemma 6 (Miyazawa *et al.* [14]). *There exists an algorithm that decides if a set of circles can be packed into a bin and its result is a set of polynomials which can give a 4α-packing for any given rational $\alpha > 0$. If the maximum number of circles that fit into a bin is constant, then this algorithm runs in constant time.*

Lemma 7 (Miyazawa *et al.* [14]). *Given a set $\mathcal{C}$ of n circles and its corresponding εh-packing into a bin of size $w \times h$, for some $\varepsilon \in \mathbb{R}_+$, we can find a feasible packing of $\mathcal{C}$ into a bin of size $w \times (1 + n\sqrt{6\varepsilon})h$ in linear time.*

Let us call the new algorithm for 2DK as VCircleKnapsack. There are two differences between it and CircleKnapsack. The first one is that circles from each set $\mathcal{S}_j$, for $j \geq 0$, will be packed into augmented bins of size $w_j \times (1+3\varepsilon)h_j$ because of the previous discussion. Thus, the grid used by VCircleKnapsack is also augmented, i.e., its subbins have size $w_j \times (1+3\varepsilon)h_j$. A consequence of this difference is that the final packing will be into an augmented bin $B_{w\times(1+3\varepsilon)h}$. The second difference is that instead of asking for an optimal solution for an instance of 2DMK, we will use an approximate solution for it, which will be given by an algorithm called MKGreedyPacking. This means that free bins generated during the execution of CircleKnapsack are not the same as the ones generated by VCircleKnapsack. We will call the set of free bins of this viable algorithm as G_j^t, for each $j \geq 0$ and some t.

Formally, MKGreedyPacking will receive as input $(\mathcal{S}_j, w_j, h_j, g, \varepsilon)$, where, to simplify notation, $g = |G_j^t|$. It will return a packing of some circles of $\mathcal{S}_j$ into g bins of augmented size $w_j \times (1+3\varepsilon)h_j$. The general idea of MKGreedyPacking is to create a new set $\mathcal{R}_j$ of circles that correspond to the circles of $\mathcal{S}_j$ with rounded radii, such that it is possible to find an optimal solution for 2DMK for input $(\mathcal{R}_j, w_j, h_j, g)$. This solution for $\mathcal{R}_j$ is then used to build a solution for $\mathcal{S}_j$ and we show that the profit decrease during this process is not so big. We formally describe this idea in the following.

First note that, by definition, the radii of the circles in $\mathcal{S}_j$ vary between $I = w_j\varepsilon^{2r-1}/2$ and $E = w_j\varepsilon/2$. Thus, at most $M = \lceil (w_jh_j)/(\pi((w_j\varepsilon^{2r-1})/2)^2)\rceil = \lceil (4\varepsilon^2)/(\pi\varepsilon^{4r})\rceil$ circles of $\mathcal{S}_j$ can be packed into a bin of type $B_{w_j\times h_j}$, a constant. Let $\varepsilon' = \varepsilon^2/(6M^2)$. We divide the interval between I and E into K subintervals such that the ℓth subinterval starts at $I\left(1 + \varepsilon'/\varepsilon\right)^{\ell-1}$ and ends at $I\left(1 + \varepsilon'/\varepsilon\right)^{\ell}$. This means that $I\left(1 + \varepsilon'/\varepsilon\right)^{K-1} \leq E$, from where we find that $K \leq \log_{(1+\varepsilon'/\varepsilon)}\left(\varepsilon/\varepsilon^{2r-1}\right) + 1$, a constant. Let $\rho_\ell = I\left(1 + \varepsilon'/\varepsilon\right)^{\ell}$ for all integer ℓ such that $0 \leq \ell < K$ and $\rho_K = E$.

We now build a set $\mathcal{R}_j$ of rounded circles of $\mathcal{S}_j$ in the following way. For each $c_i \in \mathcal{S}_j$, add to $\mathcal{R}_j$ a corresponding circle of radius ρ_ℓ if $\rho_\ell < r_i \leq \rho_{\ell+1}$. We denote by η_ℓ the amount of circles of radius ρ_ℓ that $\mathcal{R}_j$ contains. We call $\kappa = (\kappa_1, \kappa_2, \ldots, \kappa_K)$ a *configuration* if $\sum_{\ell=1}^{K} \kappa_\ell \leq M$ where κ_ℓ indicates how many circles of radius ρ_ℓ there are in κ. We say a configuration κ is *feasible* if there is a packing of all its circles into a bin of size $w_j \times h_j$. We then enumerate all configurations of $\mathcal{R}_j$, which are at most M^K, and decide, with the algorithm of Lemma 6, which ones are feasible using $\alpha = \varepsilon' h_j/4$. Now for each feasible configuration, we have a packing of its circles into a bin of size $w_j \times (1 + M\sqrt{6\varepsilon'})h_j$, which is in fact a bin of size $w_j \times (1+\varepsilon)h_j$.

Let χ be the set of all feasible configurations of $\mathcal{R}_j$ and x_κ be a variable which indicates the amount of a given configuration $\kappa \in \chi$ is to be used in a solution for 2DMK. The integer program that maximizes $\sum_{\kappa\in\chi} p(\kappa)x_\kappa$ subject to constraints $\sum_{\kappa\in\chi} \kappa_\ell x_\kappa \leq \eta_\ell$, for each $0 \leq \ell < K$, and $\sum_{\kappa\in\chi} x_\kappa \leq g$, finds a set P_j of g feasible configurations of total profit $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{R}_j, g)$. We see P_j as a set of g bins of size $w_j \times (1+\varepsilon)h_j$.

We describe next how to build a set Q_j of g bins containing circles of $\mathcal{S}_j$ based on P_j, which contains circles of $\mathcal{R}_j$. Temporarily, we just copy the packing of P_j to Q_j by replacing each circle of $\mathcal{R}_j$ with the corresponding circle in $\mathcal{S}_j$. Note that, since the circles in $\mathcal{S}_j$ are bigger, they can overlap in Q_j or surpass the borders of the bins (which are of size $w_j \times (1+\varepsilon)h_j$). However, the circles will overlap by at most twice the size of the Kth (and largest) subinterval, that is, at most $2\frac{\varepsilon'}{\varepsilon}I\left(1+\varepsilon'/\varepsilon\right)^{K-1} \leq \varepsilon' w_j = \varepsilon' h_j$. Thus, by definition, Q_j is an $\varepsilon' h_j$-packing and, according to Lemma 7, we can transform it into a valid packing into bins of size $w_j \times (1+M\sqrt{6\varepsilon'})(1+\varepsilon)h_j$, or $w_j \times (1+\varepsilon)^2 h_j$, which are in fact bins of size at most $w_j \times (1+3\varepsilon)h_j$ when $\varepsilon \leq 1$. Next claim follows directly.

Claim 3. $p(Q_j) \geq p(P_j) = \mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{R}_j, g)$.

Lemma 8 shows the relation between an optimal solution for 2DMK over input $(\mathcal{S}_j, w_j, h_j, g)$ and an optimal solution for 2DMK over input $(\mathcal{R}_j, w_j, h_j, g)$.

Lemma 8. *Given a set $\mathcal{S}_j$, a number g of bins of size $w_j \times h_j$, and a set $\mathcal{R}_j$ created from $\mathcal{S}_j$ as described above,* $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{R}_j, g) \geq (1-2\varepsilon)\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, g)$.

Proof (Sketch). We use an optimal solution for 2DMK over input $(\mathcal{S}_j, w_j, h_j, g)$ to build a solution of smaller cost for 2DMK over input $(\mathcal{R}_j, w_j, h_j, g)$, which in its turn costs less than $\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{R}_j, g)$. The result follows using the relation among the radii of circles of these two sets (each circle $c_i \in \mathcal{S}_j$ has a corresponding circle $c_i^R \in \mathcal{R}_j$ of radius ρ_ℓ, where $\rho_\ell < r_i \leq \rho_{\ell+1} = \rho_\ell\left(1+\varepsilon'/\varepsilon\right)$).

□

Theorem 4 follows from Claim 3 and Lemma 8. Lemmas 9 and 10 analyse the time complexity of MKGreedyPacking and VCircleKnapsack.

Theorem 4. *For an input $(\mathcal{S}_j, w_j, h_j, |G_j^t|, \varepsilon)$, the solution given by* MKGreedyPacking *has cost at least* $(1-2\varepsilon)\mathrm{OPT}^{\mathrm{2DMK}}_{w_j\times h_j}(\mathcal{S}_j, |G_j^t|)$.

Lemma 9. MKGreedyPacking *packs a subset of circles of $\mathcal{S}_j$ into f bins of size $w_j \times (1+3\varepsilon)h_j$ in $O(|\mathcal{S}_j|M + M^K + T_{\mathrm{ILP}})$ time, where $M = \left\lceil(4\varepsilon^2)/(\pi\varepsilon^{4r})\right\rceil = O((1/\varepsilon)^{(1/\varepsilon)})$, $K = \log_{(1+\varepsilon'/\varepsilon)}\left(\varepsilon/\varepsilon^{2r-1}\right)+1 = O((1/\varepsilon)\log_{\varepsilon^{(1/\varepsilon)}}(1/\varepsilon))$, and T_{ILP} is the polynomial-time to solve the integer program, which has at most $K+1$ restrictions and at most M^K variables.*

Proof. Creating $\mathcal{R}_j$ takes time $O(|\mathcal{S}_j|)$ because we have to check each circle of $\mathcal{S}_j$. Enumerating all configurations of $\mathcal{R}_j$ takes time $O(M^K)$ because we have at most K different radii and at most M circles can be placed into a bin. Verifying if one given configuration is valid takes constant time because M is constant (Lemma 6). Solving the integer program with at most M^K variables and at most $K + M^K$ restrictions takes polynomial time T_{ILP}, according to Lenstra [11]. Creating and fixing P_j takes time $O(Mf)$, which is an upper bound on the amount of circles in P_j^R. Considering that $f \leq |\mathcal{S}_j|$, $O(Mf) = O(M|\mathcal{S}_j|)$. □

Lemma 10. VCircleKnapsack *packs a subset of circles of $\mathcal{C}$ into a bin of size $w \times (1+3\varepsilon)h$ in $O((n+M^K+T_{\mathrm{ILP}})n/\varepsilon)$ time, where $M = \lceil (4\varepsilon^2)/(\pi\varepsilon^{4r}) \rceil = O((1/\varepsilon)^{(1/\varepsilon)})$, $K = \log_{(1+\varepsilon'/\varepsilon)} \left(\varepsilon/\varepsilon^{2r-1}\right) + 1 = O((1/\varepsilon)\log_{\varepsilon^{(1/\varepsilon)}}(1/\varepsilon))$, and T_{ILP} is the polynomial-time to solve the integer program in each call of* MKGreedyPacking*, which has at most $K + 1$ restrictions and at most M^K variables.*

Proof. The external for loop of VCircleKnapsack executes $1/\varepsilon$ times. Each iteration of such for has the running time dominated by the calls to MKGreedyPacking and the time to find new free bins.

By Lemma 9, the total time used for all the calls to MKGreedyPacking is $\sum_{j \geq 0} O(|\mathcal{S}_j|M + M^k + T_{\mathrm{ILP}}) = O(n(M^K + T_{\mathrm{ILP}}))$.

Given some $j \geq 0$, we need to find each subbin of $\tilde{\mathsf{G}}_{j+1}(B_{w\times(1+3\varepsilon)h})$ that does not intersect already packed circles from $\mathcal{S}' = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_{j-1}$. Equivalently, for each circle $c_i \in \mathcal{S}'$ already packed we can find the subbins of $\tilde{\mathsf{G}}_{j+1}(B_{w\times(1+3\varepsilon)h})$ that intersect its borders, which are a constant amount. Since $|\mathcal{S}'| \leq n$, after $O(n)$ time, for each $j \geq 0$, we can find the free bins. The result follows because we bound the total of different values for j by n. □

8 PTAS for the Two-Dimensional Knapsack for Circles

The only remaining step is to show a relation between the solutions produced by VCircleKnapcask and CircleKnapsack. This is done by Lemma 11 and Theorem 5. Theorem 6 shows our main result. Recall that the free bins kept by VCircleKnapsack are denoted as G_j^t, for each $j \geq 0$ and some t.

Lemma 11. *For all $j \geq 0$,*

$$\sum_{k=0}^{j} p(G_k^t) \geq (1 - 16\varepsilon) \sum_{k=0}^{j} \mathrm{OPT}_{w_k \times h_k}^{\mathrm{2DMK}}(\mathcal{S}_k, |F_k^t|).$$

Proof. This proof is similar to the proof of Lemma 5 and, thus, omitted. □

Theorem 5. VCircleKnapsack *packs circles from $\mathcal{C}$ into a bin of size $w \times (1 + 3\varepsilon)h$ such that their total profit is at least $(1 - O(\varepsilon))\mathrm{OPT}_{w\times h}(\mathcal{C})$.*

Proof. This follows directly from Theorem 2 and Lemma 11. □

Theorem 6. *Let $(\mathcal{C}, W, H)$ be an instance of the Two-dimensional Knapsack for Circles such that $H/W \in O(1)$ and $\mathcal{C}$ contains n circles. For any constant $0 < \epsilon \leq 1$, we can obtain, in time polynomial in n, a packing of a subset of circles from $\mathcal{C}$ into an augmented bin of size $W \times (1 + 7\epsilon)H$ such that their total profit is at least $(1 - O(\epsilon))\mathrm{OPT}_{W \times H}(\mathcal{C})$.*

Proof (Sketch). We write w, h, and ε as functions of W, H, and ϵ, respectively, such that $1/\varepsilon$ is integer multiple of 2 and $h/(w\varepsilon)$ is integer. Then we use VCircleKnapsack and analyse its result considering the original input. □

References

1. Adamaszek, A., Wiese, A.: A quasi-PTAS for the two-dimensional geometric Knapsack problem. In: Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), pp. 1491–1505. Society for Industrial and Applied Mathematics, Philadelphia (2015)
2. Christensen, H.I., Khan, A., Pokutta, S., Tetali, P.: Approximation and online algorithms for multidimensional bin packing: a survey. Comput. Sci. Rev. **24**, 63–79 (2017)
3. Demaine, E.D., Fekete, S.P., Lang, R.J.: Circle Packing for Origami Design Is Hard. A K Peters/CRC Press, Singapore (2010). pp. 609–626
4. Fishkin, A.V., Gerber, O., Jansen, K., Solis-Oba, R.: On packing squares with resource augmentation: maximizing the profit. In: Proceedings of the 2005 Australasian Symposium on Theory of Computing (CATS 2005), pp. 61–67. Australian Computer Society Inc., Darlinghurst (2005)
5. Fishkin, A.V., Gerber, O., Jansen, K., Solis-Oba, R.: Packing weighted rectangles into a square. In: Jędrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 352–363. Springer, Heidelberg (2005). https://doi.org/10.1007/11549345_31
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. Hifi, M., M'Hallah, R.: A literature review on circle and sphere packing problems: models and methodologies. Adv. Oper. Res. **2009**, 1–22 (2009)
8. Hokama, P., Miyazawa, F.K., Schouery, R.C.S.: A bounded space algorithm for online circle packing. Inf. Process. Lett. **116**(5), 337–342 (2016)
9. Jansen, K., Solis-Oba, R.: Packing squares with profits. SIAM J. Discret. Math. **26**(1), 263–279 (2012)
10. Jansen, K., Zhang, G.: Maximizing the total profit of rectangles packed into a rectangle. Algorithmica **47**(3), 323–342 (2007)
11. Lenstra, H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
12. Lodi, A., Martello, S., Monaci, M., Vigo, D.: Two-Dimensional Bin Packing Problems, pp. 107–129. Wiley, Hoboken (2013)
13. Meir, A., Moser, L.: On packing of squares and cubes. J. Comb. Theory **5**(2), 126–134 (1968)
14. Miyazawa, F.K., Pedrosa, L.L.C., Schouery, R.C.S., Sviridenko, M., Wakabayashi, Y.: Polynomial-time approximation schemes for circle and other packing problems. Algorithmica **76**(2), 536–568 (2016)
15. Szabó, P.G., Markót, M.C., Csendes, T., Specht, E., Casado, L.G., García, I.: New Approaches to Circle Packing in a Square. Springer Optimization and its Applications. Springer, New York (2007). https://doi.org/10.1007/978-0-387-45676-8

Scheduling Parallelizable Jobs Online to Maximize Throughput

Kunal Agrawal[1], Jing Li[2], Kefu Lu[1(✉)], and Benjamin Moseley[3]

[1] Washington University in St. Louis, St. Louis, MO 63130, USA
kefulu@wustl.edu
[2] New Jersey Institute of Technology, Newark, NJ 07102, USA
[3] Carnegie Mellon University, Forbes Avenue, Pittsburgh, PA 15213, USA
moseleyb@andrew.cmu.edu

Abstract. In this paper, we consider scheduling parallelizable jobs online to maximize the throughput or profit of the schedule. In particular, a set of n jobs arrive online and each job J_i arriving at time r_i has an associated function $p_i(t)$ which is the profit obtained for finishing job J_i at time $t + r_i$. Each job can have its own arbitrary non-increasing profit function. We consider the case where each job is a parallel job that can be represented as a directed acyclic graph (DAG). We give the first non-trivial results for the profit scheduling problem for DAG jobs and show $O(1)$-competitive algorithms using resource augmentation.

1 Introduction

Scheduling preemptive jobs online to meet deadlines is a fundamental problem and, consequently, the area has been extensively studied. In a typical setting, there are n jobs that arrive over time. Each job J_i arrives at time r_i, has a deadline d_i, relative deadline $D_i = d_i - r_i$ and a profit or weight p_i that is obtained if the job is completed by its deadline. The *throughput* of a schedule is the total profit of the jobs completed by their deadlines and a popular scheduling objective is to maximize the total throughput of the schedule.

In a generalization of the throughput problem, each job J_i is associated with a function $p_i(t)$ which specifies the profit obtained for finishing job J_i at $r_i + t$. It is assumed that p_i can be different for each job J_i and that the functions are arbitrary non-increasing functions. We call this problem the *general profit* scheduling problem.

In this work, we consider the throughput and general profit scheduling problems in the preemptive online setting for parallel jobs. In this setting, the *online* scheduler is only aware of the job at the time it arrives in the system, and a job is *preemptive* if it can be started, stopped, and resumed from the previous position later. We model parallel jobs as a directed acyclic graph (DAG) where each job J_i is represented as an *independent* DAG. Each node in the DAG is a sequence of instructions that are to be executed and the edges in DAG represent dependencies. A node can be executed if and only if all of its predecessors have been completed. Therefore, two nodes can potentially be executed in parallel if neither precedes the other in the DAG. In this setting, each job J_i arrives as a single independent DAG and a profit of p_i is obtained if *all* nodes of the DAG are completed by job J_i's deadline. The DAG model can represent parallel programs written in many widely used parallel languages and libraries, such as OpenMP [1], Cilk Plus [2], Intel TBB [3] and Microsoft Parallel Programming Library [4].

Both the throughput and general profit scheduling problem have been studied extensively for sequential jobs. In the simplest setting, each job J_i has work or processing time W_i to be processed on a single machine (processor). It is known that there exists a deterministic algorithm which is $O(\delta)$-competitive, where δ is the ratio of the maximum to minimum density of a job [5–8]. The *density* of job J_i is $\frac{p_i}{W_i}$ (the ratio of its profit to its work). In addition, this is the best possible result for any deterministic online algorithm even in the case where all jobs have unit profit and the goal is to complete as many jobs as possible by their deadline. In the case where the algorithm can be randomized, $\Theta(\min\{\log\delta, \log\Delta\})$ is the optimal competitive ratio [9,10]. Here Δ is the ratio of the maximum to minimum job processing time.

These strong lower bounds on the competitive ratio on any online algorithm makes it difficult to differentiate between algorithms and to discover the key algorithmic ideas that work well in practice. To overcome this challenge, the now standard form of analysis in scheduling theory is a *resource augmentation* analysis [11,12]. In a resource augmentation analysis, the algorithm is given extra resources over the adversary and the competitive ratio is bounded. A s-speed c-competitive algorithm is given a processor s times faster than the optimal solution and achieves a competitive ratio of c. The seminal scheduling resource augmentation paper considered the throughput scheduling problem and gave the best possible $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm for any fixed $\epsilon > 0$ [12].

Since this work, there has been an effort to understand and develop algorithms for more general scheduling environments and objectives. In the identical machine setting where the jobs can be scheduled on m identical parallel machines (processors), a $(1+\epsilon)$-speed $O(1)$-competitive algorithm is known for fixed $\epsilon > 0$ [13]. This has been extended to the case where the machines have speed scalable processors and the scheduler is energy aware [14]. In the related machines and unrelated machines settings, similar results have been obtained as well [15]. In [16] similar results were obtained in a distributed model.

None of this prior work consider parallel jobs. Parallel jobs modeled as DAGs have been widely considered in scheduling theory for other objectives [17–24]. There has been an extensive study in the real-time system community on how to schedule parallelizable DAG jobs by their deadlines. See [17,18,25–31] for pointers to relevant work. These works consider different (yet similar) objectives, focusing on tests to determine if a given set of reoccurring jobs can *all* be completed by their deadline, in contrast to optimizing throughput or profit.

Results: In this paper, we give the *first* non-trivial results for scheduling parallelizable DAG jobs online to maximize throughput and then we generalize these results to the general profit problem. Two important parameters in the DAG setting are the critical-path length L_i of job J_i (its execution time on an infinite number of processors) and its total work W_i (its uninterrupted execution time on a single processor). The value of $\max\{L_i, W_i/m\}$ is a lower bound on the amount of time any 1-speed scheduler takes to complete job J_i on m cores. We will focus on schedulers that are aware of the values of L_i and W_i when the job arrives, but are unaware of the internal structure of the job's DAG. That is, besides L_i and W_i, the only other information a scheduler has on a job's DAG is which nodes are currently available to execute. We call such an algorithm *semi-non-clairvoyant*—for DAG tasks, this is a reasonable model for the real world programs written in languages mentioned above since the DAG generally unfolds dynamically as the program executes. We first state a simple theorem about these schedulers.

Theorem 1. *There exists inputs where any semi-non-clairvoyant scheduler requires speed augmentation of* $2 - 1/m$ *to be* $O(1)$*-competitive for maximizing throughput.*

Roughly speaking, scheduling even a single DAG job in time smaller than $\frac{W_i - L_i}{m} + L_i$ is a hard problem even offline when the entire job structure is known in advance. This is captured by the classic problem of scheduling a precedence constrained jobs to minimize the makespan. For this problem, there is no $2 - \epsilon$ approximation assuming a variant of the unique games conjecture [32]. In particular, in Sect. 4, we will give an example DAG where any semi-non-clairvoyant scheduler will take roughly $\frac{W_i - L_i}{m} + L_i$ time to complete, while a fully clairvoyant scheduler can finish in time W_i/m. By setting the relative deadline to be $D_i = W_i/m = L_i$, every semi-non-clairvoyant scheduler will require a speed augmentation of $2 - 1/m$ to have bounded competitiveness.

With the previous theorem in place, we cannot hope for a $(1+\epsilon)$-speed $O(1)$-competitive algorithm. To circumvent this hurdle, one could hope to show $O(1)$-competitiveness by either using more resource augmentation or by making an assumption on the input. Intuitively, the hardness comes from having a relative deadline D_i close to $\max\{L_i, W_i/m\}$. In practice, this is unlikely to be the case. We show that so long as $D_i \geq (1 + \epsilon)(\frac{W_i - L_i}{m} + L_i)$ then there is a $O(\frac{1}{\epsilon^6})$-competitive algorithm.

Theorem 2. *If for every job* J_i *it is the case that* $(1 + \epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$*, then there is a* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for maximizing throughput.*

We note that this immediately implies the following corollary without any assumptions on the input.

Corollary 1. *There is a* $(2+\epsilon)$*-speed* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for maximizing throughput.*

Proof. No schedule can finish a job J_i if its relative deadline is smaller than $\max\{L_i, \frac{W_i}{m}\}$ and we may assume that no such job exists. Using this, we have that $(\frac{W_i}{m} + L_i) \leq 2D_i$. Consider transforming the problem instance giving the algorithm *and* the optimal solution together $2+\epsilon$ speed. In this case, the condition of Theorem 2 is met since we can view this as scaling the work in each node of the jobs by $2+\epsilon$. This scales the work and critical-path length by $2+\epsilon$. The corollary follows by observing that in this case we are comparing to an optimal solution with $2+\epsilon$ speed which is only stronger than comparing to an optimal solution with 1 speed. □

We note that the theorem also immediately implies the following corollary for "reasonable jobs."

Corollary 2. *There is a* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^6})$*-competitive for maximizing throughput if* $(W_i - L_i)/m + L_i \leq D_i$ *for all jobs* J_i.

This assumption on the deadlines is reasonable since, as we show in Sect. 4, there exists inputs for which even the optimal semi-non-clairvoyant scheduler has unbounded performance if the deadline is smaller.

We go on to consider the general profit scheduling problem. We first make the following assumption, which is that for all jobs J_i its general profit function satisfies $p_i(d) = p_i(x_i^*)$, where $0 < d \leq x_i^*$ for some $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. This assumption states that there is no additional benefit for completing a job J_i before time x_i^*, which is the natural generalization of our assumption in the throughput case. The function is arbitrarily decreasing otherwise. Using this, we show the following.

Theorem 3. *If for every job* J_i *it is the case that* $p_i(d) = p_i(x_i^*)$*, where* $0 < d \leq x_i^*$ *for some value of* $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$ *then there is a* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for the general profit objective.*

This gives the following corollary, just as for throughput.

Corollary 3. *There is a* $(2+\epsilon)$*-speed* $O(\frac{1}{\epsilon^6})$*-competitive algorithm for maximizing general profit.*

2 Preliminaries

In the problem considered, there is a set $\mathcal{J}$ of n jobs $\{J_1, J_2, \ldots, J_n\}$ which arrive online. The jobs are scheduled on m identical processors. Job J_i arrives at time r_i. Let $p_i(t)$ be an arbitrary non-negative non-increasing function for job J_i. The value of $p_i(t)$ is the profit obtained by completing job i at time $r_i + t$. Under some schedule, let t_i be the time it takes to complete J_i after its arrival. The goal is for the scheduler to maximize $\sum_{i \in [n]} p_i(t_i)$.

A special case of this problem is scheduling jobs with deadlines. In this problem, each job J_i has a deadline d_i and obtains a profit of p_i if it is completed by this time. In this case, we let $D_i = d_i - r_i$ be the relative deadline of the job. To make the underlying ideas of our approach clear, we will first focus on proving this case and the more general problem can be found in the Sect. 5.

Each job is represented by a Directed-Acyclic-Graph (DAG). A node in the DAG is *ready* to execute if all its predecessors have completed. A job is *completed* only when *all* nodes in the job's DAG have been processed. We assume the scheduler knows the ready nodes for a job at any point in time, but does not know the entire DAG structure a priori. Any set of ready nodes can be processed at once, but each processor can only execute one node at a time.

A DAG job has two important parameters. The total *work* W_i is the sum of the processing time of the nodes in job i's DAG. The *span* or *critical-path-length* L_i is the length of the longest path in job i's DAG, where the length of the path is the sum of the processing time of nodes on the path. To show Theorem 2 we assume that $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$ for all jobs J_i throughout the remainder of the paper.

3 Jobs with Deadlines

First, we give an algorithm and analysis proving Theorem 2 when jobs have deadlines and profits. To aid the reader, a list of notation can be found in Tables 1, 2 and 3. Throughout the proof, we let C^O denote the jobs that the optimal solution completes by their deadline and let $\left\|C^O\right\|$ denote the total profit obtained by the optimal solution. Our goal is to design a scheduler that achieves profit close to $\left\|C^O\right\|$. Throughout the proof, it will be useful to discuss the aggregate number of processors assigned to a job over all time. We define a *processor step* to be a unit of time on a single processor.

Table 1. Notations and definitions throughout the paper

Notation	Definition
OPT	Optimal schedule and also optimal objective
m	The number of processors
W_i	The total work of job J_i
L_i	The span of job J_i
D_i	Relative deadline of job J_i
r_i	The arrival time of J_i
d_i	The absolute deadline of J_i (that is, $r_i + D_i$)
$A(T, v_1, v_2)$	All jobs in T with density within the range $[v_1, v_2)$
$N(T, v_1, v_2)$	$= \sum_{J_i \in A(T,v_1,v_2)} n_i$, the total number of processors required by $A(T, v_1, v_2)$
v-dense	If Job J_i has density $v_i \geq v$
δ	$< \epsilon/2$
c	$\geq 1 + \frac{1}{\epsilon\delta}$
b	$= (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$
a	$= 1 + \frac{1+2\delta}{\epsilon - 2\delta}$

Table 2. Notations and definitions specific to jobs with deadlines

Notation	Definition
p_i	The profit of job J_i
n_i	$= \frac{(W_i - L_i)}{\frac{D_i}{1+2\delta} - L_i}$, the number of processors allocated to J_i
x_i	$= \frac{W_i - L_i}{n_i} + L_i$, the maximum execution time of J_i
v_i	$= \frac{p_i}{x_i n_i}$ the density of J_i
δ-good	Job J_i has $D_i \geq (1+2\delta)x_i$
δ-fresh	At time t, job J_i has $d_i - t \geq (1+\delta)x_i$
R	The set of jobs started by S
C	The set of jobs completed by S
U	Unfinished jobs by S (that is, $R \setminus C$)
C^O	The set of jobs completed by OPT
$\mathcal{J}$	The set of all jobs
$T_O(v, \mathcal{E})$	The total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense
$T_S(v, \mathcal{E})$	The total number of processors steps S used for executing jobs in $\mathcal{E}$ that are v-dense

Table 3. Notations and definitions specific to jobs with general profit functions

Notation	Definition
$p_i(t)$	The profit of job J_i if the job with arrival time r_i completes by $r_i + t$
n_i	$= \frac{(W_i - L_i)}{\frac{x_i^*}{1+2\delta} - L_i}$, the number of processors allocated to J_i
x_i	$= \frac{W_i - L_i}{n_i} + L_i$, the maximum execution time of J_i
v_i	$= \frac{p_i(D_i)}{x_i n_i}$ the density of J_i

3.1 Algorithm

In this section, we introduce our algorithm S. On every time step, S must decide which jobs to schedule and which ready nodes of each job to schedule. When a job J_i arrives, S calculates n_i—the number of processors "allocated" to J_i. On any time step when S decides to run J_i, it will always allocate n_i processors to J_i. In addition, since S is semi-non-clairvoyant, it is unable to distinguish between ready nodes of J_i; when it decides to allocate n_i nodes to J_i, it arbitrarily picks n_i ready nodes to execute if more than n_i nodes are ready.

We first state some observations regarding work and critical-path length.

Observation 1. *If a job J_i has all of its r ready nodes being executed by a schedule with speed s on m processors, where $r \leq m$, then the remaining critical-path length of J_i decreases at a rate of s.*

As mentioned earlier, we assume that the deadline for each job follows the condition that $(1+\epsilon)(\frac{W_i - L_i}{m} + L_i) \leq D_i$ for some positive constant ϵ.

We define the following **constants**. Let $\delta < \epsilon/2, c \geq 1 + \frac{1}{\delta\epsilon}$ and $b = (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$ be fixed constants. For each job J_i, the algorithm calculates n_i as $\frac{(W_i - L_i)}{\frac{D_i}{1+2\delta} - L_i}$. The value of n_i is the number of processors our algorithm will give to job J_i if we decide to execute J_i on some time step.

Let $x_i := \frac{W_i - L_i}{n_i} + L_i$. By Observation 1 it is the case that if n_i processors are given to job i for x_i units of time then the job will be completed regardless of the order the nodes are executed in. We will consider this to be Observation 2.

Observation 2. *Job J_i can meet its deadline if it is given n_i dedicated processors for x_i time steps in the interval $[r_i, d_i]$.*

We define the ***density*** of a job as $v_i = \frac{p_i}{x_i n_i}$. Note that this is a non-standard definition of density. We define the density as $\frac{p_i}{x_i n_i}$ instead of $\frac{p_i}{W_i}$, because we will think of job i requiring $x_i n_i$ processor steps to complete by Scheduler S. Thus, this definition of density indicates the potential profit per processor step that S can obtain by executing J_i.

The scheduler S maintains jobs that have arrived but are unfinished in two priority queues. A priority queue Q stores all the jobs that have been ***started*** by S. In the priority queue, the jobs are sorted according to the density from high to low. Another priority queue P stores all the jobs that have arrived but have not been started by S. Jobs in P are also sorted according to their densities from high to low.

Job Execution: At each time step t, S picks a set of jobs in Q to execute, in order from highest to lowest density. If a job J_i has been completed or if its absolute deadline d_i has passed ($d_i > t$), S removes the job from Q. When considering job J_i, if the number of unallocated processors is at least n_i the scheduler assigns n_i processors to J_i for execution. Otherwise, it continues on to the next job. S stops this procedure when either all jobs have been considered or when there are no remaining processors to allocate.

We introduce some notations to describe how jobs are moved from queue P to Q. A job J_i is **δ-good** if $D_i \geq (1+2\delta)x_i$. A job is **δ-fresh** at time t if $d_i - t \geq (1+\delta)x_i$. For any set T of jobs, let the set $A(T, v_1, v_2)$ contains all jobs in T with density within the range $[v_1, v_2)$. We define $N(T, v_1, v_2) = \sum_{J_i \in A(T,v_1,v_2)} n_i$. This is the total number of processors that S allocates to jobs in $A(T, v_1, v_2)$. We will say that the set of job $A(T, v_1, v_2)$ *requires* $N(T, v_1, v_2)$ processors.

Adding Jobs to Q: There are two types of events that may cause S to add a job to Q. These events occur when either a job arrives or S completes a job. When a job J_i arrives, S adds it to queue Q if it satisfies the following conditions:

(1) J_i is δ-good;
(2) For all job $J_j \in Q \cup \{J_i\}$ it is the case that $N(Q \cup \{J_i\}, v_j, cv_j) \leq bm$. In words, the total number of processors required by jobs in $Q \cup \{J_i\}$ with density in the range $[v_j, cv_j)$ is no more than bm.

If these conditions are met, then J_i is inserted into queue Q; otherwise, job J_i is inserted into queue P. When a job is added to Q, we say that the job is *started* by S.

At the completion of a job, S considers the jobs in P from highest to lowest density. S first removes all jobs with absolute deadlines that have already passed. Then S checks if a job J_i in P can be moved to queue Q by checking whether job J_i is δ-fresh and condition (2) from above. If both the conditions are met, then J_i is moved from queue P to queue Q.

Remark: Note that the Scheduler S pre-computes a fixed number of processors n_i assigned to each job, which may seem strange at first glance. This is because that n_i is approximately the minimum number of dedicated cores job J_i requires to complete by $\frac{D_i}{1+2\delta} \to D_i$, without knowing J_i's DAG structure. In addition, as long as J_i can complete by its deadline, it obtains the same profit p_i. Therefore, there is no need to complete J_i earlier by executing J_i on more dedicated cores. Moreover, by carefully assigning n_i, we are able to bound the number of processor steps spent on job J_i as shown in Lemma 3, which is critical for bounding the profit obtained by the optimal solution.

Outline of the Analysis of S: Our goal is to bound the total profit that S obtains. We first discuss some basic properties of S in Sect. 3.2. In Sect. 3.3 be bound the total profit of all the jobs S starts by the total profit of jobs that S completes. Then in Sect. 3.4 we bound the total profit of the jobs the optimal solution completes by the total profit of jobs that S starts. Putting these two together, we are able to bound the performance of S.

3.2 Properties of the Scheduler

We begin by showing some structural properties for S that we will leverage in the proof. We first bound the number of processors n_i that S will allocate to job J_i.

Lemma 1. *For every job J_i we have that $n_i \leq b^2 m$.*

Proof. By assumption we know that $D_i \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. The definition of n_i gives the following.

$$n_i = \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - L_i} \leq \frac{W_i - L_i}{\frac{1+\epsilon}{1+2\delta}(\frac{W_i - L_i}{m} + L_i) - L_i} \leq \frac{1+2\delta}{1+\epsilon} m = b^2 m$$

□

Lemma 2. *Every job J_i is δ-good, i.e. $x_i(1+2\delta) \leq D_i$.*

Proof. Note that $L_i \leq \frac{1}{1+\epsilon} D_i$ by definition. Since $n_i = \frac{W_i - L_i}{\frac{D}{1+2\delta} - L_i}$, we have $x_i(1+2\delta) = (\frac{W_i - L_i}{n_i} + L_i)(1+2\delta) = (\frac{D_i}{1+2\delta} - L_i + L_i)(1+2\delta) \leq D_i$. □

The next lemma bounds the total number of processor steps occupied by a job.

Lemma 3. *$x_i n_i \le aW_i$, where a is $1 + \frac{1+2\delta}{\epsilon-2\delta}$.*

Proof. By definition we have

$$x_i n_i = W_i - L_i + n_i L_i \le W_i + \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - L_i} L_i \le W_i + \frac{W_i - L_i}{\frac{D_i}{1+2\delta} - \frac{D_i}{1+\epsilon}} \Big(\frac{D_i}{1+\epsilon}\Big)$$
$$\le W_i + \frac{(W_i - L_i)D_i(1+2\delta)}{D_i(\epsilon - 2\delta)} \le W_i + \frac{W_i(1+2\delta)}{\epsilon - 2\delta} \le W_i\Big(1 + \frac{1+2\delta}{\epsilon - 2\delta}\Big)$$

□

Observation 3. *At any time and for any $v > 0$, the total number of processors required by all the jobs J_i that are in queue Q and have density $v \le v_i < cv$ is no more than bm, i.e. $N(Q, v_i, cv_i) \le bm$.*

Proof. Jobs are only added to queue Q when a new job arrives or a job completes. According to algorithm S, at both times, a job is only added to Q when this condition is satisfied. □

3.3 Bounding the Profit of Jobs S Completes by All Jobs Started by S

In this section, we bound the profit of jobs completed by S compared to the profit of all jobs it ever starts (adds to Q). Let R denote the set of jobs S starts (that is, the set of jobs added to queue Q). Among the jobs in R, let C be the set of jobs it completes and U be the set of jobs that are unfinished. We say job J_i (and its assigned processors) is v-**dense**, if its density $v_i \ge v$. For any set A of jobs, define $\|A\|$ as $\sum_{i \in A} p_i$, the sum of the profits of jobs in the set.

Lemma 4. *For a job $J_i \in U = R \setminus C$ that was added to queue Q but does not complete by its deadline, S must have run cv_i-dense jobs for at least δx_i time steps where J_i is in Q using at least $(1-b)m$ processors at each such time.*

Proof. Since J_i is at least δ-fresh when added to Q and it does not complete by its deadline, there are at least δx_i time steps where S is not executing J_i by Observation 2. In each of these the time steps, all the m processors are executing v_i-dense jobs.

By Observation 3, jobs in Q with density in range $[v_i, cv_i)$ require at most $N(Q, v_i, cv_i) \le bm$ processors to execute. Therefore, for each of the δx_i time steps, there are at least $(1-b)m$ processors executing cv_i-dense jobs. So the total number processor steps where cv_i-dense jobs are executing is at least $\delta x_i(1-b)m$. □

We now bound the profit of the jobs completed by their deadline under S by those started.

Lemma 5. $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta})\,\|R\|$.

Proof. We use a charging scheme with credit transfers between the jobs. We give each job $J_i \in R$ a bank account B_i. Initially, all completed jobs (in C) are given p_i credits and other jobs (in U) have 0 credit. We will transfer credits between jobs in C and jobs in U. We want to show that after the credit transfer, every job J_i in R will have $B_i \geq (\epsilon - \frac{1}{(c-1)\delta})p_i$. This implies $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta})\,\|R\|$.

Now we explain how credits are transferred. For each time step, a processor executing J_i will transfer $\frac{v_j n_j}{\delta b m}$ credits from B_i to every job J_j in queue Q that has density $v_j \leq \frac{v_i}{c}$.

For every job $J_j \in U$, Lemma 4 implies that there are at least δx_j time steps where at least $(1-b)m$ processors are executing cv_j-dense jobs. By our credit transfer strategy J_j will receive at least $\frac{v_j n_j}{\delta b m}$ credits from each processor in a time step. Therefore, the total credits J_j receives is at least

$$\delta x_j(1-b)m(\frac{v_j n_j}{\delta b m}) = v_j x_j n_j(\frac{1-b}{b}) = p_i(\frac{1-b}{b}).$$

This bounds the total amount of credit each job receives. We now show that not too much credit is transferred out of each job's account. We bound this on a job by job basis. Fix a job $J_i \in R$ and consider how many credits it transfers to other jobs during its execution. By Observation 2, we know that J_i can execute for at most x_i time steps on n_i dedicated processors before its completion.

The job J_i will transfer credit to all jobs in Q with density less than $\frac{v_i}{c}$ at any point in time where J_i is being processed. These are the jobs in $A(Q, 0, \frac{v_i}{c})$. Fix an integer $l \geq 1$ and consider the set of jobs $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ in Q that have density within the range $[\frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$. Note that the total number of processors required by them is $N(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l}) \leq bm$ by Observation 3. Knowing that a job J_j in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ has density $v_j \leq \frac{v_i}{c^l}$ by definition it is the case that the total credits that J_i gives to jobs in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ per processor assigned to J_i during any time step is at most

$$\sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} \frac{v_j n_j}{\delta b m} \leq \sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} \frac{\frac{v_i}{c^l} n_j}{\delta b m} = \frac{v_i}{\delta b m c^l} \sum_{J_j \in A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})} n_j$$
$$= \frac{v_i}{\delta b m c^l} N(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l}) \leq \frac{v_i}{\delta b m c^l} bm = \frac{v_i}{\delta c^l}.$$

This bounds the total credit transferred to jobs in $A(Q, \frac{v_i}{c^{l+1}}, \frac{v_i}{c^l})$ during a time step for each processor assigned to J_i. We sum this quantity over all $l \geq 1$ and all n_i processors assigned to i to bound the total credit transferred from job J_i during a time step. Recall that $c > 1$ by definition.

$$\frac{n_i v_i}{\delta} \sum_{l=1}^{\infty} \frac{1}{c^l} = \left(\frac{n_i v_i}{\delta}\right) \frac{\frac{1}{c}}{1 - \frac{1}{c}} = \left(\frac{n_i v_i}{\delta}\right) \frac{1}{c-1}$$

Therefore, the total credits J_i transfers to all the jobs in $A(Q, 0, \frac{v_i}{c})$ over all times it is executed is at most $(\frac{x_i n_i v_i}{\delta})\frac{1}{c-1} = \frac{p_i}{(c-1)\delta}$ due to the fact that a job will be executed for at most x_i time steps in S's schedule.

Now we put these two observations together. Each job receives at least $p_i \frac{1-b}{b}$ credit and pays at most $\frac{p_i}{(c-1)\delta}$. After the credit transfer, the credits that a job J_i has is at least

$$p_i \frac{1-b}{b} - \frac{p_i}{(c-1)\delta} = p_i(\epsilon - \frac{1}{(c-1)\delta})$$

By our setting of c, this quantity is always positive. Therefore, we conclude that $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|R\|$. □

3.4 Bounding the Profit of Jobs OPT Completes by All Jobs Started by S

In this section, we bound the profit of the jobs OPT completes by all of the jobs that S starts. Our high level goal is to first bound the total amount of time OPT spends processing jobs that S does not complete by the time S spends processing jobs. Then using this and properties of S we will be able to bound the total profit of jobs OPT completes. At a high level, this follows since S focuses on processing high density jobs and OPT and S spend a similar amount of time processing jobs. We begin by showing that if not too many processors are executing $\frac{v_i}{c}$-dense jobs then all such jobs must be currently executing.

Lemma 6. *For any density v_i and time, if there are less than $b(1-b)m$ processors executing $\frac{v_i}{c}$-dense jobs, then all $\frac{v_i}{c}$-dense jobs in queue Q are executing and $N(Q, \frac{v_i}{c}, \infty) < b(1-b)m$.*

Proof. By definition, there are at least $m - b(1-b)m > bm - b(1-b)m = b^2m$ processors executing jobs with density less than $\frac{v_i}{c}$. For the sake of contradiction, suppose there is a $\frac{v_i}{c}$-dense job J_j that is not executing by S. By Lemma 1 we know that $n_j \leq b^2m$. Therefore, J_j would have been executed by S on the b^2m processors that are executing lower density jobs, a contradiction.

Now we know all $\frac{v_i}{c}$-dense jobs in queue Q are executing. By assumption they are using less than $b(1-b)m$ processors and the lemma follows. □

In the next lemma, we show that if not too many processors are running $\frac{v_i}{c}$-dense jobs then when a job arrives or completes, the schedule S will start processing a v_i-dense job that is δ-fresh, for any density v_i (if such a job exists). In particular, the job J_j will pass condition (2) of for adding jobs to Q in the definition of S.

Lemma 7. *Fix a density v_i. At a time where a new job arrives or a job completes if there are less than $b(1-b)m$ processors executing $\frac{v_i}{c}$-dense jobs, then a δ-fresh v_i-dense job J_j (arriving or in queue P) will be added to Q by S assuming such a job J_j exists.*

Proof. By Lemma 6, we know that all $\frac{v_i}{c}$-dense jobs in queue Q are executing on less than $b(1-b)m$ processors. By Lemma 1, we know that $n_j \leq b^2m$. Therefore,

$$N(Q \cup \{J_j\}, \frac{v_i}{c}, \infty) < b(1-b)m + b^2m = bm$$

Consider any δ-fresh job J_j that is also v_i-dense. Consider any job J_k where $J_j \in A(Q \cup \{J_i\}, v_k, cv_k)$. By definition of J_j being v_i-dense it must be the case that $A(Q \cup \{J_i\}, v_k, cv_k) \subseteq A(Q \cup \{J_j\}, \frac{v_i}{c}, \infty)$. The above implies that $N(Q \cup \{J_i\}, v_k, cv_k) \leq N(Q \cup \{J_j\}, \frac{v_i}{c}, \infty) \leq bm$. Thus, the condition (2) in our algorithm is satisfied. □

For an arbitrary set of jobs $\mathcal{E}$ and any $v \geq 0$ let $T_O(v, \mathcal{E})$ denote the total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense. We similarly let $T_S(v, \mathcal{E})$ be the total number of processors steps S used for executing jobs in $\mathcal{E}$ that are v-dense over all time. Now we are ready to bound the time that OPT spends on jobs that S never adds to Q.

Lemma 8. *Consider the jobs in $\mathcal{J} \setminus R$, the jobs that are never added to Q. For all $v > 0$, $T_O(v, \mathcal{J} \setminus R) \leq \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$.*

Proof. Let $\{I_k = [s_k, e_k]\}$ be the set of maximal time intervals where at least $b(1-b)m$ processors are running $\frac{v}{c}$-dense jobs in S's schedule. Notice that by definition $\sum_{k=1}^{\infty}(e_k - s_k)b(1-b)m \leq T_S(\frac{v}{c}, \mathcal{J})$.

Consider a job in $J_i \in \mathcal{J} \setminus R$ that is both δ-good and v-dense and additionally arrives during $[s_k, s_{k+1})$. Note that during the intervals $[e_k, s_{k+1}]$, less than $b(1-b)m$ processors are executing $\frac{v}{c}$-dense jobs. Lemma 7 implies that if J_i arrives during $[e_k, s_{k+1}]$ it will be added to Q. This contradicts the assumption that $J_i \in \mathcal{J} \setminus R$. Therefore, J_i must arrive during $[s_k, e_k)$ and is in queue P at time e_k.

Note that at time e_k, the number of processors executing $\frac{v}{c}$-dense jobs decreases, so there must be a job that completes at time e_k. Again, by Lemma 7 if J_i is δ-fresh at time e_k then it will be added to Q at this time. Again, this contradicts $J_i \in \mathcal{J} \setminus R$. Thus, the only reason that S does not add J_i to Q is because J_i is not δ-fresh at time e_k. Knowing that J_i is δ-good at r_i and is not δ-fresh at e_k, we have $e_k - s_k \geq e_k - r_i \geq \delta x_i$.

At time e_k, J_i is not δ-fresh, so $d_i - e_k < (1+\delta)x_i < \frac{1+\delta}{\delta}(e_k - s_k)$.

Let K_k be the set of v-dense jobs that arrive during $[s_k, s_{k+1})$ but are not completed by S. Because OPT can only execute all jobs in K_k during $[s_k, d_i]$ on at most m processors, we get

$$T_O(v, K_k) \leq (d_i - s_k)m = ((d_i - e_k) + (e_k - s_k))m \leq \frac{1+2\delta}{\delta}(e_k - s_k)m$$

This completes the proof, as

$$T_O(v, U) = \sum_{k=1}^{\infty} T_O(v, K_k) \leq \sum_{k=1}^{\infty}(\frac{1+2\delta}{\delta})m(e_k - s_k) \leq \frac{1+2\delta}{\delta}\frac{1}{b(1-b)}T_S(\frac{v}{c}, \mathcal{J})$$

□

Using the previous lemma, we can bound the profit of jobs completed by OPT by the profit of jobs started by S.

Lemma 9.

$$\left\|C^O\right\| \leq \left(1 + (1 + \frac{1+2\delta}{\epsilon - 2\delta})(1 + \frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)\|R\|.$$

Proof. We may assume WLOG that the adversary completes all jobs it starts. First we partition C^O, the jobs that the adversary completes, into C_R^O and C_S^O where $C_S^O = C^O \cap R$, that is, our algorithm started the job at some point. The remaining jobs are placed in C_R^O. Clearly $\left\|C_S^O\right\| \leq \|R\|$. Now it remains to bound $\left\|C_R^O\right\|$.

Consider every job in C_R^O and let the set of densities of these jobs be $\{\mu_1, \mu_2, \ldots, \mu_m\}$ from high to low and for notational simplicity let $\mu_0 = \infty$ and $\mu_{m+1} = 0$. Recall the adversary completed all jobs it started. Thus for each job with density μ_i, the adversary ran the job for a corresponding W_i processor steps. Let β_i denote the number of processor steps our algorithm takes to run jobs with densities within $(\frac{\mu_{i-1}}{c}, \frac{\mu_i}{c}]$.

We have $T_O(v, \mathcal{J} \setminus R) \leq \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$ from Lemma 8 for all densities v. Equivalently for any given density v:

$$T_O(v, \mathcal{J} \setminus R) = \sum_{i=1}^{v} W_i \leq \frac{1+2\delta}{\delta b(1-b)} \sum_{i=1}^{v} \beta_i = \frac{1+2\delta}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$$

We then sum over all densities. The subtraction of densities is necessary to insure that each density is only counted a single time.

$$\sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\sum_{i=1}^{v} W_i\right) \leq \sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{v}\beta_i\right)$$

The LHS can be simplified:

$$\sum_{v=1}^{m}\left((\mu_v - \mu_{v+1})\sum_{i=1}^{v} W_i\right) = \sum_{i=1}^{m} W_i \sum_{v=i}^{m}(\mu_v - \mu_{v+1}) = \sum_{i=1}^{m} W_i(\mu_i - \mu_{m+1}) = \sum_{i=1}^{m} W_i \mu_i$$

The RHS similarly simplifies to $\frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i$, leading to the inequality that $\sum_{i=1}^{m} W_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i$. Recall that densities such as μ_i are defined by $\mu_i = \frac{p_i}{x_i n_i}$ and $x_i n_i \leq a W_i$. Therefore:

$$\sum_{i=1}^{m} W_i\mu_i = \sum_{i=1}^{m}\frac{W_i p_i}{x_i n_i} \geq \sum_{i=1}^{m}\frac{W_i p_i}{a W_i} \geq \sum_{i=1}^{m}\frac{p_i}{(1 + \frac{1+2\delta}{\epsilon-2\delta})} = \frac{1}{(1 + \frac{1+2\delta}{\epsilon-2\delta})}\left\|C_R^O\right\|$$

And also, by the definition of β_i, we know that $\sum_{i=1}^{m}\beta_i\frac{\mu_i}{c} \leq \|R\|$.

Combining these results, we get:

$$\frac{1}{(1+\frac{1+2\delta}{\epsilon-2\delta})}\left\|C_R^O\right\| \leq \sum_{i=1}^{m} W_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)}\sum_{i=1}^{m}\beta_i\mu_i \leq \frac{1+2\delta}{\delta b(1-b)}c\left\|R\right\|$$

$$\Rightarrow \left\|C_R^O\right\| \leq \left(1+\frac{1+2\delta}{\epsilon-2\delta}\right)\left(\frac{1+2\delta}{\delta b(1-b)}\right)c\left\|R\right\|$$

$$\Rightarrow \left\|C^O\right\| = \left\|C_R^O\right\| + \left\|C_S^O\right\| \leq \left(1+(1+\frac{1+2\delta}{\epsilon-2\delta})(1+\frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)\left\|R\right\|$$

□

Finally we are ready to complete the proof, bounding the profit OPT obtains by the total profit the algorithm obtains for jobs it completed.

Lemma 10.

$$\left\|C^O\right\| \leq \frac{\left(1+(1+\frac{1+2\delta}{\epsilon-2\delta})(1+\frac{1}{\epsilon\delta})\frac{1+2\delta}{\delta b(1-b)}\right)}{\epsilon-\frac{1}{(c-1)\delta}}\left\|C\right\|$$

Proof. This is just by combination of Lemmas 5 and 9. □

Therefore, we prove Theorem 2 by showing that scheduler S is $O(\frac{1}{\epsilon^6})$-competitive for jobs with deadlines and profits, when $(1+\epsilon)(\frac{W_i-L_i}{m}+L_i) \leq D_i$.

4 Examples

In this section, we will give some example DAGs to show why Theorem 2 is close to the best theorem we can hope for using two examples. The first example, shown in Fig. 1, shows the limitations of semi-non-clairvoyance. In particular, a semi-non-clairvoyant scheduler does not know the structure of the DAG in advance since the DAG unfolds dynamically. At any time step, the scheduler only knows the ready nodes available for execution. Given this limitation, consider the DAG shown in Fig. 1. This job has one sequential chain with length $L = \frac{W}{m}$, where W is the total work of the job and m is the number of processors. The remaining $W - W/m$ work are fully parallelizable in a block and can also be done in parallel with the chain. Therefore, L is the span of the jobs.

Since a semi-non-clairvoyant scheduler cannot distinguish between ready nodes, it may make unlucky choices and execute the entire block of $W - W/m = W - L$ ready nodes first in $(W-L)/m$ time steps and then execute the chain of L nodes sequentially—leading to a total time of $(W-L)/m + L$. On the other hand, a fully clairvoyant scheduler can execute the entire DAG in W/m time. Therefore, a semi-non-clairvoyant scheduler needs at least $2 - 1/m$ speed augmentation to ensure that it can complete the DAG at the same time as OPT.

We now show another example DAG indicating that it would be reasonable to always set deadlines as $D \geq (W-L)/m + L$ if we do not know the structure of the DAG a priori. Figure 2 shows an example DAG, which consists of a chain of $L - \epsilon$ nodes followed by $W - L + \epsilon$ nodes that can run in parallel. Each node in the DAG takes ϵ time to run, so the total work of the DAG is W and the span is L. For such a DAG, even a fully clairvoyant scheduler needs $L-\epsilon+\frac{W-L+\epsilon}{m} = \frac{W-L}{m}+L-\epsilon(1-\frac{1}{m})$, which approaches to $\frac{W-L}{m}+L$ when $\epsilon \to 0$.

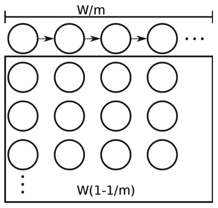


Fig. 1. Example 1

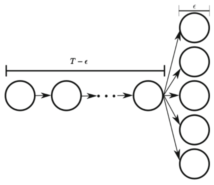


Fig. 2. Example 2

5 Jobs with General Profit Functions

In this section, we focus on a more general case. In particular, each job J_i has a non-negative non-increasing profit function $p_i(t)$ indicating its profit if the job with arrival time r_i completes by $r_i + t$. Our goal is to design a scheduler that maximizes the profit to make it close to what the optimal solution can obtain, denoted as $\|O\|$.

First, we present our scheduler S parameterized using a fixed constant $0 < \epsilon < 1$. Similar to Sect. 3.1, let $\delta < \epsilon/2$, $c \geq 1 + \frac{1}{\delta\epsilon}$ and $b = (\frac{1+2\delta}{1+\epsilon})^{1/2} < 1$ be fixed constants.

Upon the arrival of a job J_i, the scheduler S assigns a number of allocated cores n_i, a relative deadline D_i and a set of time steps I_i to J_i (according to the assignment procedure described below). In each time step t in I_i, we say that J_i is assigned to t. Scheduler S always executes the highest density jobs that is assigned to t. If S decides to execute J_i in a time step, it will give n_i processors to J_i. Let $x_i := \frac{W_i - L_i}{n_i} + L_i$. We define the ***density*** of a job as $v_i = \frac{p_i(D_i)}{x_i n_i} = \frac{p_i(D_i)}{W_i + (n_i - 1)L_i}$. We now formally specify the algorithm of scheduler S for job assignment and execution.

Assigning cores, deadlines and slots to jobs: When a job J_i arrives, the scheduler will assign a relative deadline D_i and a set of time steps I_i with n_i processors. These time steps are the only time steps in which J_i is allowed to run.

Recall (from Theorem 3) that we assume that the profit function stays the same until some value $x_i^* \geq (\frac{W_i - L_i}{m} + L_i)(1 + \epsilon)$. The number of assigned processors n_i is calculated as $n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i}$. The assignment for D_i is determined by searching all the potential deadlines D to find the minimum valid deadline. The set of time steps I_i is determined using the chosen deadline D_i.

For each ***potential relative deadline*** $D > (1 + \epsilon)L_i$, scheduler S checks whether it is a valid deadline by the following steps. First, it selects a set of time steps I. Assuming D is assigned to J_i, then the density of J_i is $v = \frac{p_i(D)}{W_i + (n_i - 1)L_i}$. For each time step t from r_i to $r_i + D$, let $\|I(t)\|$ be the number of time steps that have already been added to I before considering time step t. Let $J(t)$ denote the set of jobs that are currently has time t among its assignments. We only add t to the set I if it satisfies the following condition: For every job $J_j \in J(t)$, it is the case that $N\left(J(t) \cup \{J_i\}, v_j, cv_j\right) \leq bm$. In words, the total number of processors required by jobs in $J(t) \cup \{J_i\}$ with density in the range $[v_j, cv_j)$ is no more than bm.

I contains all the time steps during $[r_i, r_i + D_i)$ that can be assigned to J_i. If $\|I\| \geq (1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right)$, which is at least δ times longer than the time J_i required to run on n_i processors, then the deadline D is said to be *valid*. Note that a valid assignment always exists by setting the deadline large enough.

Among all the valid assignments, S chooses the smallest valid deadline for J_i, which results in the highest profit. Given this deadline D_i, J_i will be assigned with the corresponding set I_i. Because D_i is the minimum valid deadline, the corresponding set I_i must satisfy $\|I_i\| = (1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right)$; otherwise, there must exist a shorter deadline D that is also valid. Intuitively, with this assignment, J_i can complete by its deadline if no other jobs interfere. Note that J_i may not be completed by its deadline as we will allow higher density jobs that arrive after J_i to be scheduled during I_i.

Executing jobs: At each time step t, S picks a set of jobs in $J(t)$ to execute in order from highest to lowest density, where $J(t)$ are the set of jobs that have been assigned to time step t. That is, jobs J_i where $t \in I_i$. When considering job J_i, if the number of unallocated processors is at least n_i, then the scheduler allocates n_i processors to J_i. Otherwise, it continues on to the next job in $J(t)$. S stops this procedure when either all jobs have been considered or when there are no remaining processors to allocate.

Remark: Unlike the scheduler for jobs with deadlines, here we try to complete a job J_i by a calculated deadline D_i that is as close to x_i^* as possible. This is because the obtained profit decreases as the completion time increases but there is no additional benefit for completing a job J_i before time x_i^*. With a carefully designed deadline D_i, we are able to prove the performance bound of the scheduler. Similarly to Sect. 3, we start by stating the basic properties of the scheduler S, followed by bounding the total profit obtained by S. However, the proofs that bound the profit of jobs that are completed by OPT differ greatly from that for jobs with deadlines. This is because in addition to losing the profit of jobs that do not complete by their assigned deadlines, scheduler S can also loses profit compared to OPT if the completion time of a job under S is later than under OPT. By taking into account all these jobs, we are able to bound the performance of S for jobs with general profit functions.

5.1 Properties of the Scheduler

We begin by showing some structural properties for S that we will leverage in the proof and can be obtained directly from the algorithm of scheduler S. Note that these lemmas are similar to the lemmas shown in Sect. 3.2 if we replace x_i* with D_i. We state them here again for completeness.

Lemma 11. *For every job J_i we have that $n_i \leq b^2 m$, where $b = (\frac{1+2\delta}{1+\epsilon})^{1/2}$.*

Proof. By definition, we know that $x_i^* \geq (1+\epsilon)(\frac{W_i - L_i}{m} + L_i)$. Therefore, we have

$$n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i} \leq \frac{W_i - L_i}{\frac{1+\epsilon}{1+2\delta}(\frac{W_i - L_i}{m} + L_i) - L_i} \leq \frac{1+2\delta}{1+\epsilon} m = b^2 m$$

□

Lemma 12. *Under scheduler S, we have $x_i n_i \leq a W_i$ and $v_i \geq \frac{p_i(D_i)}{aW_i}$, where $a = 1 + \frac{1+2\delta}{\epsilon - 2\delta}$.*

Proof. By definition, $x_i^* > L_i(1+\epsilon)$. Therefore, we have

$$\begin{aligned} x_i n_i &= W_i - L_i + n_i L_i = W_i + \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - L_i} L_i \leq W_i + \frac{W_i - L_i}{\frac{x_i^*}{1+2\delta} - \frac{x_i^*}{1+\epsilon}} \Big(\frac{x_i^*}{1+\epsilon}\Big) \\ &\leq W_i + \frac{(W_i - L_i) x_i^* (1+2\delta)}{x_i^* (\epsilon - 2\delta)} \leq W_i \Big(1 + \frac{1+2\delta}{\epsilon - 2\delta}\Big) \end{aligned}$$

Therefore, we have $v_i = \frac{p_i(D_i)}{x_i n_i} \geq \frac{p_i(D_i)}{aW_i}$. □

Lemma 13. *For every job J_i with the assignment n_i, D_i and I_i, Job J_i can meet its deadline D_i, if it is executed by S for at least x_i time steps in I_i (on n_i dedicated processors).*

Lemma 14. *For every job J_i, $x_i(1+2\delta) \leq x_i^*$.*

Proof. Note that $L_i \leq \frac{1}{1+\epsilon} D_i$ by requirement of potential assignment. Since $n_i = \frac{W_i - L_i}{\frac{x_i^*}{1+\epsilon} - L_i}$, we have $x_i(1+2\delta) = (\frac{W_i - L_i}{n_i} + L_i)(1+2\delta) \leq (\frac{x_i^*}{1+\epsilon} - L_i + L_i)(1+2\delta) = \frac{x_i^*}{1+\epsilon}(1+2\delta) \leq x_i^*$. □

Lemma 15. *At any time step t during the execution and for any density range $[v, cv)$, the total number of cores required by all the jobs $J_i \in J(t)$ (that have been assigned to t) with density $v \leq v_i < cv$ is no more than bm, i.e. $N(J(t), v_i, cv_i) \leq bm$.*

5.2 Bounding the Profit of Jobs S Completes

Similar to Sect. 3.3, we bound the profit of jobs completed by scheduler S compared to the profit of all jobs. Let $\mathcal{J}$ denote the set of jobs arrived during the execution, C denote the set of jobs that actually complete before their deadlines assigned by S, and $U = \mathcal{J} \setminus C$ be the set of jobs that didn't finish by their deadlines assigned by S. We say job J_i (and its assigned processors during execution) is **v-dense**, if its density $v_i \geq v$. For any set A of jobs, define $\|A\|$ as $\sum_{J_i \in A} p_i(D_i)$, the sum of the profits of jobs in the set under S.

Lemma 16. *For a job $J_i \in \mathcal{J} \setminus C$ that does not complete by its deadline, the number of time steps in I_i where S runs cv_i-dense jobs using at least $(1-b)m$ processors is at least δx_i.*

Proof. From Lemma 13, we know that job J_i can complete if it can execute for x_i time steps by S. Also note that according to the assignment process $(1+\delta)x_i = \|I_i\|$, where $\|I_i\|$ is the number of time steps assigned to J_i during $[r_i, r_i + D_i]$. Since it does not complete by its deadline, there are at least δx_i time steps in I_i where S does not execute J_i. Consider each of these time steps t. According to Lemma 15, jobs in $J(t)$ with density in range $[v_i, cv_i)$ require at most $N(J(t), v_i, cv_i) \leq bm$ processors to execute. Therefore, there must be at least $(1-b)m$ processors executing cv_i-dense jobs. Otherwise, S would execute all jobs in $A(J(t), v_i, cv_i)$, which includes job J_i. □

Lemma 17. $\|C\| \geq (\epsilon - \frac{1}{(c-1)\delta}) \|\mathcal{J}\|$.

The proof is similar to that of Lemma 5 and is omitted for brevity.

5.3 Bounding the Profit of Jobs OPT Completes

Similar to Sect. 3.4, we will now bound the profit of the jobs OPT completes. We are first going to consider the number of processor steps OPT spends on jobs that S finishes later than OPT. For these jobs, we assume that S makes no profit since the profit function may become 0 as soon as OPT finishes it. Our high level goal is to first bound the total number of processor steps OPT spends on these jobs, which will allow us to bound OPT's profit. This section of the proof differ greatly from the throughput case.

We begin by showing that if not too many processors are executing $\frac{v_i}{c}$-dense jobs then all such jobs must be currently processed under S.

Lemma 18. *Consider a job J_i and a time $t^* < D_i$. For any time step $t \in [r_i, r_i + t^*] \backslash I_i$ (that is not added to I_i by S), the total number of processors required by $\frac{v_i}{c}$-dense jobs in $J(t)$ must be more than $b(1-b)m$, i.e., $N(J(t), \frac{v_i}{c}, \infty) > b(1-b)m$.*

Proof. Because $t \in [r_i, r_i + t^*] \setminus I_i$ and $t^* < D_i$, we know that time step t is before D_i.

Since t is not added to I_i, it must be the case that for some density $v_j \in (\frac{v_i}{c}, v_i]$, the required condition is not true, i.e., $N(J(t) \cup \{J_i\}, v_j, cv_j) > bm$. Note that v_j must be in the range $(\frac{v_i}{c}, v_i]$. This is because without assigning J_i to time step t it is true that $N(J(t), v_j, cv_j) \leq bm$ according to S, therefore J_i must have a density within the range of $[v_j, cv_j)$ in order to make impact.

By Lemma 11, we know that $n_i \leq b^2 m$. Thus, we have

$$N(J(t), v_j, cv_j) = N(J(t) \cup \{J_i\}, v_j, cv_j) - n_i > bm - b^2 m = b(1-b)m$$

Therefore, we obtain $N(J(t), \frac{v_i}{c}, \infty) \geq N(J(t), v_j, cv_j) > b(1-b)m$. □

Let O be the set of jobs completed by OPT. For each job $J_i \in O$, let d be the difference between J_i's completion time and arrival time under OPT; the profit of J_i under OPT is $p_i(d)$. According to the assumption in Theorem 3, we know that if $d \leq x_i^*$, then $p_i(d) = p_i(x_i^*)$ for some $x_i^* \geq (\frac{W_i - L_i}{m} + L_i)(1+\epsilon)$. Therefore, we can assume that OPT assigns a relative deadline D_i^* to J_i, where $D_i^* = \max\{d, x_i^*\}$. Thus, OPT obtains a profit of $p_i(d) = p_i(D_i^*)$.

Lemma 19. *Consider a job J_i such that D_i assigned by scheduler S is larger than the deadline D_i^* assigned by* OPT*, i.e., $D_i > D_i^*$, the number of time steps during $[r_i, r_i + D_i^*)$ where scheduler S is actively executing $\frac{v_i}{c}$-dense jobs on at least $b(1-b)m$ cores is at least $\frac{\delta}{1+2\delta}D_i^*$.*

Proof. By definition of D_i^* and Lemma 14, we know that $D_i^* \geq x_i^*$.

Consider the number of time steps in time interval $[r_i, r_i + D_i^*]$ that are added to I_i, it must be less than $(1+\delta)\left(\frac{W_i - L_i}{n_i} + L_i\right) = (1+\delta)x_i$; otherwise, D_i^* would be a valid deadline under scheduler S with higher profit. Therefore, the number of time steps in $[r_i, r_i + D_i^*] \setminus I_i$ is more than $D_i^* - (1+\delta)x_i \geq D_i^* - \frac{1+\delta}{1+2\delta}x_i^* \geq D_i^* - \frac{1+\delta}{1+2\delta}D_i^* = \frac{\delta}{1+2\delta}D_i^*$.

By Lemma 18, we know that for each time step $t \in [r_i, r_i + D_i^*] \setminus I_i$, the total number of processors required by $\frac{v_i}{c}$-dense jobs in $J(t)$ must be more than $b(1-b)m$. Therefore, there must be at least $b(1-b)m$ cores executing $\frac{v_i}{c}$-dense jobs under scheduler S at time step t and the number of such steps is at least $\frac{\delta}{1+2\delta}D_i^*$. □

Among the jobs in O, let O_1 be the set of jobs that the deadline D_i assigned by scheduler S is no larger than that assigned by OPT, i.e., $D_i \leq D_i^* < \infty$. In other words, the obtained profit of these jobs under scheduler S is no less than that under OPT, i.e., $p_i(D_i) \geq p_i(D_i^*)$, since the profit function $p_i(t)$ is non-increasing. Let O_2 be the remaining jobs $O_2 = O \setminus O_1$. Let $\|X\|^*$ be the total profit that OPT obtains from jobs in X and $\|X\|$ be the total profit that S obtains from jobs in X. For jobs in O_1, we have $\|O_1\|^* \leq \|O_1\|$.

For an arbitrary set of jobs $\mathcal{E}$ and any $v \geq 0$ let $T_O(v, \mathcal{E})$ denote the total work processed by the optimal schedule for the jobs in $\mathcal{E}$ that are v-dense. Let β_i denote the total number of time steps where S is actively processing job J_i. By definition, we have $\beta_i \leq \frac{x_i}{1+\epsilon}$. We similarly let $T_S(v, \mathcal{E})$ be the summation of $\beta_i n_i$ over all jobs i in $\mathcal{E}$ that are v-dense. Note that this counts the total number of processor steps S executes jobs in $\mathcal{E}$ that are v-dense over all time.

Now we are ready to bound the time that OPT spends on jobs O_2 that scheduler S obtains less profit than OPT.

Lemma 20. *Consider a job J_i in O_2, the deadline D_i assigned by scheduler S is longer than deadline D_i^* assigned by* OPT*. For all $v > 0$, $T_O(v, O_2) \leq \frac{2(1+2\delta)}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$.*

Proof. For any job $J_i \in O_2$, we denote the lifetime of J_i under OPT as the time interval $[r_i, r_i + D_i^*)$, where D_i^* is the deadline assigned by OPT. For any density $v > 0$, let l be the number of time steps of the union of the lifetimes of all jobs in $A(O_2, v, \infty)$. By definition, $T_O(v, O_2) \leq lm$, since OPT can execute them on at most m processors.

Let $M \subseteq O_2$ be the minimum subset of O_2 that the union of the lifetimes of jobs in M covers the same time intervals of jobs in O_2. By the minimality of M, we know that at any time t, there are at most two jobs in M that cover time t. Therefore, we can further partition M into two sets M_1 and M_2, where for any two jobs in M_1 or any two jobs in M_2, their lifetimes do not overlap. By definition, either M_1 or M_2 has a union lifetime that is at least $l/2$ and we assume WLOG it is M_1.

Consider $J_i \in M_1$ and let k_i be the number of time steps during its lifetime $[r_i, r_i + D_i^*)$ where scheduler S is actively executing $\frac{v_i}{c}$-dense jobs on at least $b(1-b)m$ cores. By Lemma 19, we know $k \geq \frac{\delta}{1+2\delta} D_i^*$. Therefore, during $[r_i, r_i + D_i^*)$ the number of processor steps where S is processing $\frac{v_i}{c}$-dense jobs is at least $b(1-b)m\frac{\delta}{1+2\delta} D_i^*$.

Let $K = \sum_{M_1} k_i$, be the total number of processor steps where S is processing $\frac{v}{c}$-dense jobs (since $v_i \geq v$) during the intervals in M_1. Thus, by definition,

$$K \geq \frac{\delta b(1-b)}{1+2\delta} m \sum_{J_i \in M_1} D_i^* > \frac{\delta b(1-b)}{1+2\delta} m \times \frac{l}{2} \geq \frac{\delta b(1-b)}{2(1+2\delta)} T_O(v, O_2)$$

Clearly, by adding additional intervals that are not in M_1, we have $T_S(\frac{v}{c}, \mathcal{J}) \geq K > \frac{\delta b(1-b)}{2(1+2\delta)} T_O(v, O_2)$, which gives us the bound. □

Lemma 21.

$$\|O\|^* = \|O_1\|^* + \|O_2\|^* \leq \left(1 + (1 + \frac{1+2\delta}{\epsilon - 2\delta})(1 + \frac{1}{\epsilon\delta})\frac{2(1+2\delta)}{\delta b(1-b)}\right) \|\mathcal{J}\|$$

Proof. First, by the definition of O_1 and O_2, we have $\|O\|^* = \|O_1\|^* + \|O_2\|^*$ and $\|O_1\|^* \leq \|O_1\| \leq \|\mathcal{J}\|$. Now it remains to bound $\|O_2\|$.

We have $T_O(v, O_2) \leq \frac{2(1+2\delta)}{\delta b(1-b)} T_S(\frac{v}{c}, \mathcal{J})$ from Lemma 20 for all densities v. The remaining proof for the lemma is similar to that in Lemma 9, except for a different constant. Therefore, $\|O_2\|^* \leq (1 + \frac{1+2\delta}{\epsilon - 2\delta}) c \frac{2(1+2\delta)}{\delta b(1-b)} \|\mathcal{J}\|$. Taking the summation of $\|O_1\|^* + \|O_2\|^*$ completes the proof. □

Finally we are ready to complete the proof, bounding the profit OPT obtains by the total profit the algorithm obtains for jobs it completed.

Lemma 22. $\|C^O\| \leq \frac{1 + ac\frac{2(1+2\delta)}{\delta b(1-b)}}{\epsilon - \frac{1}{(c-1)\delta}} \|C\|$.

Proof. This is just by combination of Lemmas 17 and 21. □

6 Conclusion

Scheduling jobs online to maximize throughput is a fundamental problem, yet there has been little study of this topic when jobs are parallelizable and represented as DAGs. We give the first non-trivial result showing that a scheduling algorithm is provably good for maximizing throughput. In addition, we extend the result and give an algorithm for the general profit scheduling problem with DAG jobs.

There are several directions for future work. First, we want to design and implement more practical schedulers that have similar theoretical performance but are work-conserving and require fewer preemptions. Second, in this paper we focus on semi-non-clairvoyant algorithms that do not have any knowledge of the internal structure of the DAG. This lets us to provide very general results. However, it is possible that by using the internal structure one could design algorithms with better performance for some special DAG structures. Finally, we are also interested in exploring whether fully non-clairvoyant algorithms can have comparable performance for throughput.

References

1. OpenMP: OpenMP Application Program Interface v4.0, July 2013. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
2. Intel: Intel CilkPlus, September 2013. https://www.cilkplus.org/
3. Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. O'Reilly Media, Inc., Sebastopol (2010)
4. Campbell, C., Miller, A.: A Parallel Programming with Microsoft Visual C++: Design Patterns for Decomposition and Coordination on Multicore Architectures. Microsoft Press, Redmond (2011)
5. Baruah, S.K., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L.E., Shasha, D., Wang, F.: On the competitiveness of on-line real-time task scheduling. Real-Time Syst. **4**(2), 125–144 (1992)
6. Baruah, S.K., Koren, G., Mishra, B., Raghunathan, A., Rosier, L.E., Shasha, D., Wang, F.: On-line scheduling in the presence of overload. In: Symposium on Foundations of Computer Science, pp. 100–110 (1991)
7. Koren, G., Shasha, D.: Dover: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. **24**(2), 318–339 (1995)
8. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. Theor. Comput. Sci. **130**(1), 5–16 (1994)
9. Kalyanasundaram, B., Pruhs, K.: Fault-tolerant real-time scheduling. Algorithmica **28**(1), 125–144 (2000)
10. Koren, G., Shasha, D.: MOCA: a multiprocessor on-line competitive algorithm for real-time system scheduling. Theor. Comput. Sci. **128**(1&2), 75–97 (1994)
11. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
12. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4), 617–643 (2000)
13. Bansal, N., Chan, H.-L., Pruhs, K.: Competitive algorithms for due date scheduling. Algorithmica **59**(4), 569–582 (2011)
14. Pruhs, K., Stein, C.: How to schedule when you have to buy your energy. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX/RANDOM 2010. LNCS, vol. 6302, pp. 352–365. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15369-3_27
15. Im, S., Moseley, B.: General profit scheduling and the power of migration on heterogeneous machines. In: Symposium on Parallelism in Algorithms and Architectures (2016)
16. Lucier, B., Menache, I., Naor, J., Yaniv, J.: Efficient online scheduling for deadline-sensitive jobs: extended abstract. In: 25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2013, pp. 305–314 (2013)

17. Saifullah, A., Ferry, D., Li, J., Agrawal, K., Lu, C., Gill, C.D.: Parallel real-time scheduling of dags. IEEE Trans. Parallel Distrib. Syst. **25**(12), 3242–3252 (2014)
18. Li, J., Chen, J.-J., Agrawal, K., Lu, C., Gill, C.D., Saifullah, A.: Analysis of federated and global scheduling for parallel real-time tasks. In: ECRTS 2014, pp. 85–96 (2014)
19. Agrawal, K., He, Y., Hsu, W.J., Leiserson, C.E.: Adaptive task scheduling with parallelism feedback. In: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) (2006)
20. Agrawal, K., He, Y., Leiserson, C.E.: Adaptive work stealing with parallelism feedback. In: Proceedings of the Annual ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), March 2007
21. He, Y., Hsu, W.-J., Leiserson, C.E.: Provably efficient online non-clairvoyant adaptive scheduling. In: IPDPS (2007)
22. Ma, L., Chamberlain, R.D., Agrawal, K.: Performance modeling for highly-threaded many-core GPUs. In: Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 84–91, June 2014
23. Agrawal, K., Li, J., Lu, K., Moseley, B.: Scheduling parallel DAG jobs online to minimize average flow time. In: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, pp. 176–189 (2016)
24. Robert, J., Schabanel, N.: Non-clairvoyant scheduling with precedence constraints. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 491–500 (2008)
25. Baruah, S.: Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In: 26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, 8–11 July 2014, pp. 97–105 (2014)
26. Baruah, S.: Federated scheduling of sporadic DAG task systems. In: 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, 25–29 May 2015, pp. 179–186 (2015)
27. Baruah, S.: The federated scheduling of systems of conditional sporadic DAG tasks. In: 2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, 4–9 October 2015, pp. 1–10 (2015)
28. Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A.: The global EDF scheduling of systems of conditional sporadic DAG tasks. In: 27th Euromicro Conference on Real-Time Systems, ECRTS 2015, pp. 222–231 (2015)
29. Baruah, S.: The federated scheduling of constrained-deadline sporadic DAG task systems. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, pp. 1323–1328 (2015)
30. Li, J., Agrawal, K., Lu, C., Gill, C.: Analysis of global EDF for parallel tasks. In: ECRTS (2013)
31. Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S., Wiese, A.: Feasibility analysis in the sporadic DAG task model. In: ECRTS (2013)
32. Svensson, O.: Conditional hardness of precedence constrained scheduling on identical machines. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, pp. 745–754 (2010)

Reactive Proximity Data Structures for Graphs

David Eppstein, Michael T. Goodrich, and Nil Mamano(✉)

Department of Computer Science, University of California, Irvine, USA
{eppstein,goodrich,nmamano}@uci.edu

Abstract. We consider data structures for graphs where we maintain a subset of the nodes called *sites*, and allow proximity queries, such as asking for the closest site to a query node, and update operations that *enable* or *disable* nodes as sites. We refer to a data structure that can efficiently react to such updates as *reactive*. We present novel reactive proximity data structures for graphs of polynomial expansion, i.e., the class of graphs with small separators, such as planar graphs and road networks. Our data structures can be used directly in several logistical problems and geographic information systems dealing with real-time data, such as emergency dispatching. We experimentally compare our data structure to Dijkstra's algorithm in a system emulating random queries in a real road network.

1 Introduction

Proximity data structures are well-known in computational geometry [10], where sites are points in the plane and distance is measured, e.g., by the Euclidean metric. In this paper, we are interested in proximity data structures for graphs, where sites are defined by a distinguished subset of the vertices and distance is measured by shortest-path distance in the graph. That is, we assume a graph, G, is given and fixed (like a road network for a geographic region) and that distance is measured by shortest paths in this graph. With respect to updates, a non-distinguished vertex in G can be *enabled* to become a site or an existing site can be *disabled* to no longer be a site, and we want our data structure to *react* to such updates so as to be able to quickly answer proximity queries, such as nearest-neighbor or closest-pair queries for sites.

Definition 1 (Reactive proximity). *Given a set, U, known as the* universe, *and a distance function, $d(*)$, for elements in U, maintain a subset, $P \subseteq U$, of* sites, *allowing the following operations:*

– ***Proximity Queries. nearest-neighbor:*** *given a query element $q \in U$, return a site p in P minimizing $d(q,p)$;* ***closest-pair:*** *return a pair, $p, q \in P$, minimizing $d(p,q)$;* ***bichromatic-closest-pair:*** *suppose we have $P = R \cup B$, where $R \cap B = \emptyset$. Then, return a pair, $p, q \in P$, minimizing $d(p,q)$, such that $p \in R$ and $q \in B$.*

– ***Updates. enable:*** *add an element from U to P;* ***disable:*** *remove an element from P.*

We refer to data structures that can support such queries and updates as *reactive* data structures,[1] in that we have a fixed universe of objects which can be enabled or disabled, and we need to quickly *react* to such events as updates. In this paper, we study the case where the set U is the set of nodes of a graph, and d is the shortest-path distance.

1.1 Applications

There are a number of interesting applications for reactive proximity data structures for graphs, including several logistical problems in geographic information systems dealing with real-time data. Consider, for instance, an application to connect drivers and clients in a private-car service, such as Uber or Lyft, or even a future driverless car service. The data structure could maintain the set of cars waiting at various locations in a city to be put into service. When a client requires a driver, she queries the data structure to find the car nearest to her. This car is then disabled (i.e., it is no longer available) until it completes the trip for this client, at which point the car is then enabled (i.e., it is available) at this new location. Alternatively, we could consider a similar application in the context of police or emergency dispatching, where the data structure maintains the locations of a set of available first responder vehicles. In Sect. 3, we experiment with this type of system emulating random queries in a real road network.

As another example dealing with geo-spatial data, which the authors explore in a companion paper [22], suppose we are given a set of sites representing the locations of certain facilities, such as post offices or voting locations. We wish to partition the vertices of the graph into geographic regions, one for each facility, such that each region has a specified size (in number of nodes) and the shapes of the regions satisfy certain compactness criteria. As we show in the companion paper, a greedy matching algorithm can exploit an efficient reactive data structure to quickly build such a partitioning of the graph.

Reactive proximity data structures can also be useful in other domains, such as content distribution networks, like the one maintained by Akamai. For instance, the data structure could maintain the set of nodes that contain a certain file of interest, like a movie. When another node in the network needs this information, the data structure could be used to find the closest node that can transfer it. Updates allow us to model how copies of such a file migrate in the network, e.g., for load balancing, so that we enable a node when it gets a copy of the file and disable a node when it passes it to another server.

[1] We also call such data structures *reactive* to distinguish the kinds of updates we allow (changes to the subset of distinguished vertices) from *dynamic* data structures in which the structure of the graph itself can change, e.g., by vertex or edge insertions or deletions.

Moreover, having a reactive proximity data structure for graphs could allow us to design interesting algorithms that rely on the existence of such data structures. For instance, we discuss how to use them to solve the geometric stable roommates problem [2] in Sect. 2.

1.2 Our Results

In this paper we present a family of reactive data structures for answering proximity problems in graphs. The data structures we present all use a technique based on graph separator hierarchies and they apply to any graphs for which such hierarchies can be built. A *separator* in a given n-vertex graph is a subset of nodes such that removing it partitions the remaining graph into two disjoint subgraphs, each of size at most $2n/3$. The *size* of a separator is the number of nodes in this subset. A *separator hierarchy* is the result of recursively subdividing a graph by using separators. It has been shown recently that classes of graphs with separators of size $O(n^c)$, for any $c < 1$, coincide with the classes of graphs of polynomial expansion [15]. Thus, any graph in this family is suitable for our data structures.

Graphs of polynomial expansion are sparse, but some sparse graphs (such as bounded degree expanders) do not have polynomial expansion. Nonetheless, many important sparse graph families have polynomial expansion. One of the first classes that was shown to have sublinear separators is the class of planar graphs, which have $O(n^{0.5})$-size separators [36]. Separators of the same asymptotic size have also been proven to exist in k-planar graphs [14], bounded-genus graphs [29], minor-closed graph families [34], graphs with sparse crossing graphs [25], and the graphs of certain four-dimensional polyhedra [20].

While road networks are not quite planar because of bridges and underpasses, experimental results show that they can be modeled by graphs with sparse crossing graphs. These graphs, like planar graphs, have $O(n^{0.5})$-size separators [25]. This is fortunate, because it allows our data structures to be used in geographic information systems, e.g., for real-world road networks, as we explore experimentally in this paper.

Not surprisingly, the main technical challenge faced in designing efficient reactive proximity data structures for graphs lies in their reactive nature. In a variant where the sites (i.e., the nodes in P) are fixed, there is a well known solution: the graph-based Voronoi diagram, which maintains the closest site to each node in the graph [26]. With this information, queries can be answered in constant time. However, the Voronoi diagram is not easy to update, requiring $O(n \log n)$ time in sparse graphs with n nodes, which is the same time as for creating the diagram from scratch. If we optimize for update time instead, we could avoid maintaining any information and answer queries directly using a shortest-path algorithm from the query node. Updates would take constant time; queries could be answered using Dijkstra's algorithm [9], which runs in $O(n \log n)$ time in sparse graphs and in $O(n)$ time for graphs with a known separator hierarchy [32]. However, this is clearly not a good solution either if we want fast query times.

Our solutions amount to finding a "sweet spot" between these two extremes. Our novel data structure for the reactive nearest-neighbor problem supports queries in $O(n^{0.5})$ time and updates in $O(n^{0.5} \log \log n)$ time when the underlying graph is a planar graph or a road network. More generally, if the graph belongs to a family with separators of size $S(n) = O(n^c)$, for some $0 < c < 1$, queries and updates run in $O(S(n))$ and $O(S(n) \log \log n)$ time, respectively. Moreover, since forests have separators of size one, the data structure can be shown to have $O(\log n)$ query time and $O(\log n \log \log n)$ update time when the underlying graph is a forest or, more generally, when it has bounded treewidth.

1.3 Prior Related Work

Knuth's discussion of the classic post office problem has given rise to a long line of research on structures for spatial partitioning for answering nearest-neighbor queries, including an entire literature on the topic of Voronoi diagrams [3,8]. Given a collection of sites, these diagrams partition a space into regions such that the points in each region have a particular site as their nearest. Furthermore, Erwig [26] shows that Voronoi diagrams can be extended to graphs and that such structures can be constructed using Dijkstra's shortest-path algorithm (e.g., see [9,31]). These Voronoi diagram structures are efficient for instances when the set of sites is fixed, but they tend to perform poorly for cases in which sites can be inserted or removed. Such *dynamic nearest neighbor* problems have been addressed in geometric settings. For exact two-dimensional dynamic nearest neighbors, a data structure with $O(n^\epsilon)$ update and query time was given by Agarwal *et al.* [1], and improved to $O(\log^6 n)$ by Chan [6], and to $O(\log^5 n)$ by Kaplan *et al.* [33]. Because of the high complexities of these methods, researchers have also looked at finding approximate nearest neighbors in dynamic point sets. For example, dynamic versions of quadtrees and k-d trees are known [38], and the skip-quadtree data structure can answer approximate nearest neighbor queries and updates in logarithmic time [23].

The graph-based variant of the dynamic nearest neighbor problem that we study falls into the area of *dynamic graph algorithms*, the subject of extensive study [21]. There has been much research on shortest paths in dynamic graphs, e.g., see [4,5,12,13,35,37]. However, previous work has primarily focused on edge insertion and deletion updates rather than the vertex enable/disable updates that we study. Exceptions are the work of Eppstein on maintaining a dynamic subset of vertices in a sparse graph and keeping track of whether it is a dominating set [19], and the work of Italiano and Frigioni on dynamic connectivity for subsets of vertices in a planar graph [28], but these are very different problems from the proximity problems that we study here. To the best of our knowledge, no one has considered maintaining nearest-neighbor data structures for graphs subject to enabling and disabling of sites.

Separator hierarchies, the main technique used in this paper, have proven useful for solving many graph problems [27,30]. Of these, the most related to our problem is the $O(n)$-time single-source shortest path problem for planar graphs when edge weights are non-negative [32]. The same algorithm applies more generally to graphs that have $O(n^{0.5})$-size separators and for which the separator hierarchy can be built in $O(n)$ time.

2 Reactive Nearest-Neighbor Data Structure for Graphs

We consider undirected graphs with nonnegative edge weights. We begin by describing the concept of a separator hierarchy.

Recall that a *separator* in a given n-vertex graph is a subset $\mathcal{S}$ of nodes such that the removal of $\mathcal{S}$ (and its incident edges) partitions the remaining graph into two disjoint subgraphs (with no edges from one to the other), each of size at most $2n/3$. It is allowed for these subgraphs to be disconnected; that is, removing $\mathcal{S}$ can partition the remaining graph into more than two connected components, as long as those components can be grouped into two subgraphs that are each of size at most $2n/3$. A *separator hierarchy* is the result of recursively subdividing a graph by using separators. Note that since children have size at most $2/3$ the size of the parent, the separator hierarchy is a binary tree of $O(\log n)$ height.

Small separators are necessary for the efficiency of our data structure. As we mentioned, the analysis will depend on the size of the separators for a particular graph family, which we denote by $S(n)$, and which we assume to be of the form n^c with $0 < c < 1$ (because, e.g., forests have separators of size one, and that changes the analysis).

The reactive nearest-neighbor data structure that we describe in this section yields the following theorem.

Theorem 1. *Given an n-node graph from a graph family with separators of size $S(n) = n^c$, with $0 < c < 1$, which can be constructed in $O(n)$ time, it is possible to initialize a reactive data structure that uses $O(nS(n))$ space, in $O(nS(n))$ time, that answers nearest-neighbor queries in $O(S(n))$ time and updates in $O(S(n)\log n)$ time. Alternatively, the data structure could have $O(nS(n)\log n)$ initialization time, $O(S(n))$ query time, and $O(S(n)\log\log n)$ update time.*

2.1 Preprocessing

Initially, we are given the graph $G = (V, E)$ and the subset $P \subseteq V$ of sites. The creation of our data structure consists of two phases. The first phase does not depend on the choice of the subset P of sites, while the second phase incorporates our knowledge of P into the data structure. Note that there are two kinds of nodes of interest: separator nodes and sites. The two sets may intersect, but should not be confused.

Site-independent phase. First, we build a *separator hierarchy* of the graph. This hierarchy can be constructed in $O(n)$ time and space in planar graphs [30] and road networks [25].
Second, we compute, for each graph in the hierarchy, the distance from each separator node to every other node. This computation can be represented as a collection of single-source shortest-path problems, one for each separator node.

As we already mentioned, each single-source shortest path problem can be solved in $O(n)$ time in graphs with $O(n^{0.5})$-size separators and for which we can build a separator hierarchy in linear time [32]. Therefore, in such graphs the time to compute shortest-path distances at the top level of the hierarchy is $O(nS(n))$. We can analyze the time to compute these distances at all levels of the hierarchy by the recurrence

$$T(n) = T(x) + T(y) + O(nS(n)),$$

where x and y are the sizes of two subgraphs, chosen so that $x + y \leq n$, $\max(x, y) \leq 2n/3$, and (among x and y obeying these constraints) so that $T(x) + T(y)$ is maximum. The recurrence is dominated by its top-level $O(nS(n))$ term, and has a solution that is also $O(nS(n))$.

Site-dependent phase. For each graph H in the separator hierarchy, and each separator node s in H, we initialize a priority queue Q_s. The elements stored in Q_s are the sites that belong to H, and their priorities are their distances from s. If we implement the priority queue as a binary heap, constructing each queue Q_s takes linear time. Thus, the time at the top level of the hierarchy is linear per separator node, and the total time analysis of this phase is $O(nS(n))$ as before.

Adding the space and time for the two phases together gives $O(nS(n))$ for planar graphs, or for other graphs on which we can use the linear-time separator-hierarchy-based shortest path algorithm. If we instead use the simpler Dijkstra's algorithm to compute shortest paths, the running time becomes the slightly slower bound of $O(nS(n) \log n)$, plus the time to build the separator hierarchy.

2.2 Queries

Given a node q, we find two sites: the closest site to q with (a) paths restricted to the same side of the partition as q, and with (b) paths containing at least one separator node. The paths considered in both cases cover all possible paths, so one of the two found sites will be the overall closest site to q.

- To find the site satisfying condition (a), we can relay the query to the subgraph of the separator hierarchy containing q. This satisfies the invariant that the query node is a node of the graph. This case does not arise if q is a separator node.
- To find the site satisfying condition (b), we need the shortest path from q to any site, but only among paths containing separator nodes. Note that if a shortest path goes through a separator s, it should end at the site closest to s. Therefore, the length of the shortest path starting at q, going through s, and ending at any site, is $d(q, s) + d(s, \min(Q_s))$, where $\min(Q_s)$ denotes the element with the smallest key in Q_s. We can find the site satisfying condition (b) by considering all the separator nodes and retaining the one minimizing this sum.

The time to find paths of type (b) is $O(S(n))$, since there are $O(S(n))$ separator nodes to check and each takes constant time, as we precomputed all the needed distances. Therefore, the time to find all paths of types (a) and (b) can be analyzed by the recurrence

$$T(n) \leq T(2n/3) + O(S(n)),$$

where the $T(2n/3)$ bound dominates the actual time for recursing in a single subgraph of the separator hierarchy. The solution to this recurrence is $O(S(n))$, when $S(n)$ is polynomial, and therefore the time per query is $O(S(n))$ in this case. If $S(n)$ is constant or polylogarithmic, then the query time is $O(S(n) \log n)$.

We can also implement a heuristic optimization for queries so that we do not need to check every separator node to find a node satisfying condition (b). At each graph of the separator hierarchy, we can sort, for each node, all the separators by distance. This increases the space used by the data structure by a constant factor. Then, after obtaining the recursive candidate satisfying condition (a), to find the second candidate, we consider the separator nodes in order by distance to the query node q. Suppose p is the closest site we have found so far. As soon as we reach a separator node s such that $d(q, s) \geq d(q, p)$, we can stop and ignore the rest of separator nodes, since any site reached through them would be further from q than p. In our experiments (Sect. 3), this optimization reduced the average query runtime by a factor between 1.5 and 9.5, depending on the number of sites. It is more effective when there are many sites, as then the closest site will tend to be closer than many separators at the upper levels of the hierarchy.

2.3 Updates

Suppose that we wish to enable or disable a node p from the set of sites P. Note that, when we perform such an update, the structures computed during the site-independent preprocessing phase (the separator hierarchy and the computation of distances) do not change, as they do not depend on the choice of P. However, we will need to update Q_s for every separator node s in the top-level separator $\mathcal{S}$, by adding or removing p (according to whether we are adding or removing it from P).

Moreover, if p is not a separator node, it will belong to one of the two recursive subgraphs in the separator hierarchy. In this case, we also need to update the data structure for the subgraph containing p recursively, since p will appear in the priority queues of the separator nodes in that subgraph.

The time to add or remove p in all top-level priority queues is $O(\log n)$ per priority queue, for a total time of $O(S(n) \log n)$ at the top level. Again, if we formulate and solve a recurrence for the running time at all levels of the separator hierarchy, this time will be dominated by the top level time, giving a total time of $O(S(n) \log n)$ per update.

We can also obtain an asymptotically faster update time of $O(S(n) \log \log n)$ by, for each separator vertex s, replacing the distances from s to all other nodes by the ranks of these distances in the sorted list of distances. That is, if the set of distances in sorted order from s to the other nodes are

$$d_1, d_2, d_3, \dots$$

with $d_1 < d_2 < d_3 < \cdots$, we could replace these numbers by the numbers

$$1, 2, 3, \dots$$

without changing the comparison between any two distances. This replacement would allow us to use a faster integer priority queue, such as a van Emde Boas tree [16], in place of the binary heap representation of each priority queue Q. However, in order to use this optimization, we need to add the time to sort the distances in the preprocessing time, which increases to $O(nS(n)\log n)$ (assuming a comparison sort is used).

2.4 Additional Applications

We remark that our dynamic nearest neighbor data structure can be extended to directed graphs. The only required change is to compute distances *from* and *to* every separator node. To obtain the latter, we can use Dijkstra's algorithm in the reverse graph.

In addition, the conga line data structure of Eppstein [18] solves the dynamic closest-pair problem with $O(\log n)$ query time, $O(T(n)\log n)$ insertion time, and $O(T(n)\log^2 n)$ deletion time, where $T(n)$ is the time per operation (query or update) of a dynamic nearest-neighbor data structure. Therefore, by combining the data structure in this paper with the conga line data structure, we obtain a data structure for the reactive closest pair problem in graphs with separators of size $S(n) = O(n^c)$, with $0 < c < 1$, that achieves $O(\log n)$ query time, $O(S(n)\log n \log\log n)$ enable time, and $O(S(n)\log^2 n \log\log n)$ disable time.

Furthermore, by combining the data structure in this paper with the data structure from [17], we obtain the same running times for the reactive bichromatic closest-pair problem as for the reactive closest-pair problem. Not using our data structure would result in linear or super-linear times for either queries or updates.

The nearest-neighbor data structure can also be used for the metric stable roommates problem, extending the work of Arkin *et al.* [2] to graphs. In the original problem, we wish to match a set of points in a geometric space so that there is no unmatched pair, (p, q), such that p and q are both closer to each other than the points they are matched to. For points in general position, they show that a simple greedy algorithm, which repeatedly matches and removes a closest-pair of points, will produce a solution to the geometric stable roommates problem. An efficient algorithm for this problem is the nearest-neighbor chain algorithm, which solves it in $O(|P|)$ queries and updates of a reactive nearest-neighbor data structure [22]. Hence, combined with our data structure, we can solve the greedy matching problem (for metric stable roommates in graphs) in $O(nS(n) + |P|S(n)\log n)$ time. Without our data structure, using a shortest-path algorithm in the nearest-neighbor chain algorithm increases this time to $\Omega(|P|n)$.

Finally, note that forests and graphs with bounded treewidth have separators of size O(1), e.g., see [7]. If we reformulate and solve the recurrence equations accounting for the fact that there is constant number of separator nodes, we obtain an $O(n \log n)$ preprocessing time, $O(\log n)$ query time, and $O(\log^2 n)$ update time (or $O(\log n \log \log n)$ using an integer priority queue as discussed above).

3 Experiments

In this section, we evaluate our data structure empirically on real-world road network data, the Delaware road network from the DIMACS dataset [11]. We consider the biggest connected component of the network, which has 48812 nodes and 60027 edges. This dataset has been planarized: overpasses and underpasses have been replaced by artificial intersection nodes. Each trial in our experiment begins with a number of uniformly distributed random sites, and then performs 1000 operations. We consider the cases of only queries, only updates, and a mixture of both (see Fig. 1). The updates alternate between enables and disables, whereas the operations in the mixed case alternate between queries and updates. We compare the performance of our data structure against a basic data structure that simply uses Dijkstra's algorithm for the queries.

3.1 Implementation Details

We implemented the algorithms in Java 8.[2] We then executed them and timed them as run on an Intel Core CPU i7-3537U 2.00 GHz with 4 GB of RAM, under Windows 10.

We implemented the optimization for queries described in Sect. 2.2, and compared it with the unoptimized version in order to evaluate if its worth the extra space. For updates, we used a normal binary heap, as these tend to perform better in practice than more sophisticated data structures.

A factor that affects the efficiency of the data structure is the size and balance of the separators. Our hierarchy for the Delaware road network had a total of 504639 nodes across 8960 graphs up to 13 levels deep. Among these graphs, the biggest separator had 81 nodes. Rather than implementing a full planar separator algorithm to find the separators (recall that the data had been planarized), we choose the smallest of two simply-determined separators: the vertical and horizontal lines partitioning the nodes into two equal subsets. While these are not guaranteed to have size $O(\sqrt{n})$, past experiments on the transversal complexity in road networks [24] indicate that straight-line traversals of road networks should provide separators with low complexity, making it unnecessary to incorporate the extra complexity of a full planar graph separator algorithm.

When a separator partitions a graph in more than two connected components, we made one child per component. Thus, our hierarchy is not necessarily a binary tree, and may be shallower. We set the base case size to 20. At the base case, we perform Dijkstra's algorithm. Experiments with different base-case sizes did not affect the performance significantly.

[2] The source code is available at github.com/nmamano/NearestNeighborInGraphs.

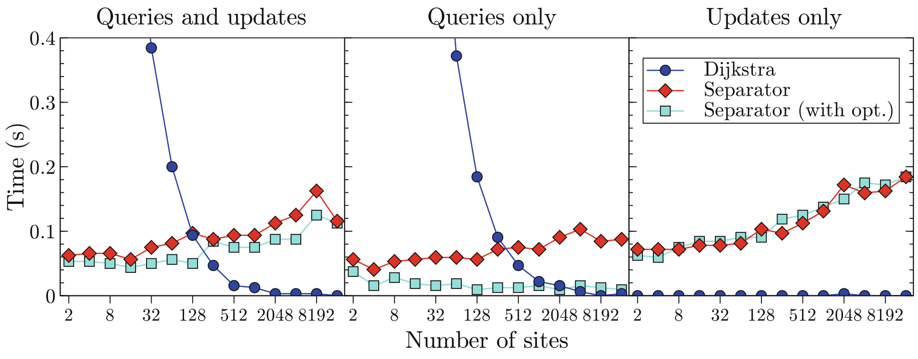


Fig. 1. Time needed by the data structures to complete 1000 operations in the Delaware road network [11] for a range of number of sites (in a logarithmic scale). Each data point is the average of 5 runs with different sets of random sites (the same sets for all the algorithms).

3.2 Results

Figure 1 depicts the results. Table 1 shows the corresponding data for the case of mixed operations, which is the case of interest in a reactive model.

- The runtime of Dijkstra's algorithm is roughly proportional to the number of sites, because with more sites it requires less exploration to find the closest one. Moreover, initialization and updates require virtually no time. Thus, this choice is superior for large numbers of sites, while being orders of magnitude slower when the number of sites is low (see Table 1).
- The data structure based on a separator hierarchy is not affected as much by the number of sites. The update runtime only increases slightly with more sites because of the operations with bigger heaps. This is consistent with its asymptotic runtime, which is $O(S(n) \log n)$ for any number of sites. The optimization, which reduces the number of separators needed to be checked, can be seen to have a significant effect on queries, especially as the number of sites increases: it is up to 9.5 times faster on average with 2048 sites. However, since it has no effect in updates, in the mixed model with the same number of updates and queries the improvement is less significant.
- The data structure requires a significant amount of time to construct the hierarchy. Our code constructed the hierarchy for the Delaware road network in around 15 s. Fortunately, this hierarchy only needs to be built once per road network. The limiting factor is the space requirement of $O(n\sqrt{n})$, which caused us to run out of memory for other road networks from the DIMACS dataset with over 10^5 nodes.

Table 1. Time in milliseconds needed by the data structures to complete 1000 operations (mixed queries and updates) in the Delaware road network for a range of number of sites (in a logarithmic scale). Each data point is the average, minimum, and maximum, of 5 runs with different sets of random sites (the same sets for all the algorithms).

# sites	Dijkstra	Separator	Separator (with opt.)
2	3797 (3672 – 3906)	63 (47 – 94)	53 (31 – 94)
4	2303 (2203 – 2359)	66 (63 – 78)	53 (47 – 63)
8	1272 (1250 – 1297)	66 (47 – 78)	50 (47 – 63)
16	694 (641 – 781)	56 (47 – 63)	44 (31 – 47)
32	384 (359 – 406)	75 (63 – 94)	50 (47 – 63)
64	200 (172 – 219)	81 (63 – 94)	56 (47 – 63)
128	94 (94 – 94)	97 (94 – 109)	50 (47 – 63)
256	47 (47 – 47)	88 (78 – 94)	84 (78 – 109)
512	16 (16 – 16)	94 (94 – 94)	75 (63 – 78)
1024	13 (0 – 16)	94 (94 – 94)	75 (63 – 78)
2048	3 (0 – 16)	113 (94 – 125)	88 (78 – 94)
4096	3 (0 – 16)	125 (109 – 156)	88 (78 – 94)
8192	3 (0 – 16)	163 (125 – 188)	125 (94 – 156)
16384	0 (0 – 0)	116 (109 – 125)	113 (94 – 156)

4 Conclusion

We have studied reactive proximity problems in graphs, giving a family of data structures for such problems. While we have focused on applications in geographic systems dealing with real-time data, the problem is primitive enough that it seems likely that it will arise in other domains of graph theory, such as network protocols. We would like to explore other applications in the future.

As we discussed in Sect. 3.1, a big factor in the runtime of any data structure based on separator hierarchies is the choice of separators. It may be of interest to compare the benefits of a simpler but lower-quality separator construction algorithm versus a slower and more complicated but higher-quality separator construction algorithm in future experiments.

References

1. Agarwal, P.K., Eppstein, D., Matoušek, J.: Dynamic half-space reporting, geometric optimization, and minimum spanning trees. In: 33rd Symposium Foundations of Computer Science (FOCS), pp. 80–89 (1992)
2. Arkin, E.M., Bae, S.W., Efrat, A., Okamoto, K., Mitchell, J.S., Polishchuk, V.: Geometric stable roommates. Inf. Process. Lett. **109**(4), 219–224 (2009). http://www.sciencedirect.com/science/article/pii/S0020019008003098

3. Aurenhammer, F.: Voronoi diagrams–a survey of a fundamental geometric data structure. ACM Comput. Surv. **23**(3), 345–405 (1991)
4. Buriol, L.S., Resende, M.G.C., Thorup, M.: Speeding up dynamic shortest-path algorithms. INFORMS J. Comput. **20**(2), 191–204 (2008)
5. Chan, E.P.F., Yang, Y.: Shortest path tree computation in dynamic graphs. IEEE Trans. Comput. **58**(4), 541–557 (2009)
6. Chan, T.M.: A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. J. ACM **57**(3), 16:1–16:15 (2010)
7. Chung, F.R.K.: Separator theorems and their applications. In: Paths, flows, and VLSI-layout (Bonn, 1988), Algorithms Combin. vol. 9, pp. 17–34. Springer, Berlin (1990)
8. Clarkson, K.L.: Nearest-neighbor searching and metric space dimensions. In: Shakhnarovich, G., Darrell, T., Indyk, P. (eds.) Nearest-Neighbor Methods in Learning and Vision: Theory and Practice, pp. 15–59. MIT Press (2006). Chapter 2
9. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York City (2001)
10. De Berg, M., Cheong, O., Van Kreveld, M., Overmars, M.: Computational Geometry: Introduction. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
11. Demetrescu, C., Goldberg, A.V., Johnson, D.S.: 9th DIMACS implementation challenge: shortest paths (2006). http://www.dis.uniroma1.it/~challenge9/
12. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. ACM **51**(6), 968–992 (2004)
13. Djidjev, H.N., Pantziou, G.E., Zaroliagis, C.D.: On-line and dynamic algorithms for shortest path problems. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 193–204. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59042-0_73
14. Dujmović, V., Eppstein, D., Wood, D.R.: Structure of graphs with locally restricted crossings. SIAM J. Discrete Math. **31**(2), 805–824 (2017)
15. Dvořák, Z., Norin, S.: Strongly sublinear separators and polynomial expansion. SIAM J. Discrete Math. **30**(2), 1095–1101 (2016)
16. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time. In: 16th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 75–84 (1975)
17. Eppstein, D.: Dynamic Euclidean minimum spanning trees and extrema of binary functions. Discrete Comput. Geom. **13**(1), 111–122 (1995)
18. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs. J. Exp. Algorithmics **5** (2000)
19. Eppstein, D.: All maximal independent sets and dynamic dominance for sparse graphs. ACM Trans. Algorithms **5**(4), Article No. 38 (2009)
20. Eppstein, D.: Treetopes and their graphs. In: 27th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 969–984 (2016). http://dl.acm.org/citation.cfm?id=2884435.2884504
21. Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Atallah, M.J. (ed.) Algorithms and Theory of Computation Handbook, pp. 9.1–9.28, 2nd edn. CRC Press(2010). http://www.info.uniroma2.it/~italiano/Papers/dyn-survey.ps.Z
22. Eppstein, D., Goodrich, M.T., Korkmaz, D., Mamano, N.: Defining equitable geographic districts in road networks via stable matching. In: 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (2017)

23. Eppstein, D., Goodrich, M.T., Sun, J.Z.: Skip quadtrees: dynamic data structures for multidimensional point sets. Int. J. Comput. Geom. Appl. **18**(1–2), 131–160 (2008)
24. Eppstein, D., Goodrich, M.T., Trott, L.: Going off-road: transversal complexity in road networks. In: 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 23–32 (2009)
25. Eppstein, D., Gupta, S.: Crossing patterns in nonplanar road networks. In: 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, September 2017
26. Erwig, M.: The graph Voronoi diagram with applications. Networks **36**(3), 156–163 (2000)
27. Frieze, A.M., Miller, G.L., Teng, S.H.: Separator based parallel divide and conquer in computational geometry. In: 4th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 420–429 (1992). http://doi.acm.org/10.1145/140901.141934
28. Frigioni, D., Italiano, G.F.: Dynamically switching vertices in planar graphs. Algorithmica **28**(1), 76–103 (2000)
29. Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A separator theorem for graphs of bounded genus. J. Algorithms **5**(3), 391–407 (1984)
30. Goodrich, M.T.: Planar separators and parallel polygon triangulation. J. Comput. Syst. Sci. **51**(3), 374–389 (1995)
31. Goodrich, M.T., Tamassia, R.: Algorithm Design and Applications, 1st edn. Wiley, Hoboken (2014)
32. Henzinger, M.R., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. J. Comput. Syst. Sci. **55**(1), 3–23 (1997)
33. Kaplan, H., Mulzer, W., Roditty, L., Seiferth, P., Sharir, M.: Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In: 28th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 2495–2504 (2017)
34. Kawarabayashi, K., Reed, B.: A separator theorem in minor-closed classes. In: 51st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 153–162 (2010)
35. King, V.: Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: 40th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 81–89 (1999)
36. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
37. Roditty, L., Zwick, U.: On dynamic shortest paths problems. Algorithmica **61**(2), 389–401 (2011)
38. Samet, H.: The design and analysis of spatial data structures. Addison-Wesley Series in Computer Science. Addison-Wesley, Reading (1990)

Mutants and Residents with Different Connection Graphs in the Moran Process

Themistoklis Melissourgos[1(✉)], Sotiris Nikoletseas[2,3], Christoforos Raptopoulos[2,3], and Paul Spirakis[1,2,3]

[1] Department of Computer Science, University of Liverpool, Liverpool, UK
{T.Melissourgos,P.Spirakis}@liverpool.ac.uk
[2] Computer Technology Institute and Press "Diophantus" (CTI), Patras, Greece
Nikole@cti.gr
[3] Computer Engineering and Informatics Department, University of Patras, Patras, Greece
Raptopox@ceid.upatras.gr

Abstract. The Moran process, as studied by Lieberman et al. [10], is a stochastic process modeling the spread of genetic mutations in populations. In this process, agents of a two-type population (i.e. mutants and residents) are associated with the vertices of a graph. Initially, only one vertex chosen uniformly at random (u.a.r.) is a mutant, with fitness $r > 0$, while all other individuals are residents, with fitness 1. In every step, an individual is chosen with probability proportional to its fitness, and its state (mutant or resident) is passed on to a neighbor which is chosen u.a.r. In this paper, we introduce and study for the first time a generalization of the model of [10] by assuming that different types of individuals perceive the population through different graphs defined on the same vertex set, namely $G_R = (V, E_R)$ for residents and $G_M = (V, E_M)$ for mutants. In this model, we study the fixation probability, namely the probability that eventually only mutants remain in the population, for various pairs of graphs.

In particular, in the first part of the paper, we examine how known results from the original single-graph model of [10] can be transferred to our 2-graph model. In that direction, by using a Markov chain abstraction, we provide a generalization of the Isothermal Theorem of [10], that gives sufficient conditions for a pair of graphs to have fixation probability equal to the fixation probability of a pair of cliques; this corresponds to the absorption probability of a birth-death process with forward bias r.

In the second part of the paper, we give a 2-player strategic game view of the process where player payoffs correspond to fixation and/or extinction probabilities. In this setting, we attempt to identify best responses for each player. We give evidence that the clique is the most beneficial graph for both players, by proving bounds on the fixation probability when one of the two graphs is complete and the other graph belongs to various natural graph classes.

P. Spirakis—The work of this author was partially supported by the ERC Project ALGAME and the EPSRC Project EP/P020372/1 "Algorithmic Aspects of Temporal Graphs".

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 790–804, 2018.
https://doi.org/10.1007/978-3-319-77404-6_57

In the final part of the paper, we examine the possibility of efficient approximation of the fixation probability. Interestingly, we show that there is a pair of graphs for which the fixation probability is exponentially small. This implies that the fixation probability in the general case of an arbitrary pair of graphs cannot be approximated via a method similar to [2]. Nevertheless, we prove that, in the special case when the mutant graph is complete, an efficient approximation of the fixation probability is possible through an FPRAS which we describe.

Keywords: Moran process · Fixation probability
Evolutionary dynamics

1 Introduction

The Moran process [14] models antagonism between two species whose critical difference in terms of adaptation is their *relative fitness*. A *resident* has relative fitness 1 and a *mutant* relative fitness $r > 0$. Many settings in Evolutionary Game Theory consider fitness as a measure of reproductive success; for examples see [3, 7, 15]. A generalization of the Moran process by Lieberman et al. [10] considered the situation where the replication of an individual's fitness depends on some given structure, i.e. a directed graph. This model gave rise to an extensive line of works in Computer Science, initiated by Mertzios and Spirakis in [12].

In this work we further extend the model of [10] to capture the situation where, instead of one given underlying graph, each species has its own graph that determines their way of spreading their offsprings. As we will show, due to the process' restrictions only one species will remain in the population eventually. Our setting is by definition an interaction between two players (species) that want to maximize their probability of occupying the whole population.

This strategic interaction is described by an 1-sum bimatrix game, where each player (resident or mutant) has all the strongly connected digraphs on n nodes as her pure strategies. The resident's payoff is the *extinction probability* and the mutant's payoff is the *fixation probability*. The general question that interests us is: what are the pure Nash equilibria of this game (if any)? To gain a better understanding of the behaviour of the competing graphs, we investigate the best responses of the resident to the clique graph of the mutant.

This model and question is motivated by many interesting problems from various, seemingly unrelated scientific areas. Some of them are: idea/rumor spreading, where the probability of spreading depends on the kind of idea/rumor; computer networks, where the probability that a message/malware will cover a set of terminals depends on the message/malware; and also spread of mutations, where the probability of a mutation occupying the whole population of cells depends on the mutation. Using the latter application as an analogue for the rest, we give the following example to elaborate on the natural meaning of this process.

Imagine a population of identical somatic *resident* cells (e.g. biological tissue) that carry out a specific function (e.g. an organ). The cells connect with each other in a certain way; i.e., when a cell reproduces it replaces another from a specified set of candidates, that is, the set of cells connected to it. Reproduction here is the replication of the genetic code to the descendant, i.e. the hardwired commands which determine how well the cell will adapt to its environment, what its chances of reproduction are and which candidate cells it will be able to reproduce on.

The changes in the information carried by the genetic code, i.e. mutations, give or take away survival or reproductive abilities. A bad case of mutation is a cancer cell whose genes force it to reproduce relentlessly, whereas a good one could be a cell with enhanced functionality. A mutation can affect the cell's ability to adapt to the environment, which translates to chances of reproduction, or/and change the set of candidates in the population that should pay the price for its reproduction.

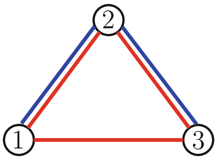
Now back to our population of resident cells which, as we said, connect with each other in a particular way. After lots of reproductions a mutant version of it shows up due to replication mistakes, environmental conditions, etc. This *mutant* has the ability to reproduce in a different rate, and also, to be connected with a set of cells different than the one of its resident version. For the sake of argument, we study the most pessimistic case, i.e. our mutant is an extremely aggressive type of cancer with increased reproduction rate and maximum unpredictability; it can replicate on any other cell and do that faster than a resident cell. We consider the following motivating question: Supposing this single mutant will appear at some point in time on a random cell equiprobably, what is the best structure (network) of our resident cells such that the probability of the mutant taking over the whole population is minimized?

The above process that we informally described captures the real-life process remarkably well. As a matter of fact, a mutation that affects the aforementioned characteristics in a real population of somatic cells occurs rarely compared to the time it needs to conquer the population or get extinct. Therefore, a second mutation is extremely rare to happen before the first one has reached one of those two outcomes and this allows us to study only one type of mutant per process. In addition, apart from the different reproduction rate, a mutation can lead to a different "expansionary policy" of the cell, something that has been overlooked so far.

2 Definitions

Each of the population's individuals is represented by a label $i \in \{1, 2, \ldots, n\}$ and can have one of two possible types: R (*resident*) and M (*mutant*). We denote the *set of nodes* by V, with $n = |V|$, and the *set of resident (mutant) edges* by $E_R(E_M)$. The node connections are represented by directed edges; A node i has a *type R(M) directed edge* $(ij)_R((ij)_M)$ towards node j if and only if when i is chosen and is of type $R(M)$ then it can reproduce on j with positive probability. The aforementioned components define two directed graphs; the *resident graph* $G_R = (V, E_R)$ and the *mutant graph* $G_M = (V, E_M)$. A node's type determines its fitness; residents have *relative fitness* 1, while mutants have relative fitness $r > 0$.

Our process works as follows: We start with the whole population as residents, except for one node which is selected uniformly at random to be mutant. We consider discrete time, and in each time-step an individual is picked with probability proportional to its fitness, and copies itself on an individual connected to it in the corresponding graph (G_R or G_M) with probability determined by the (weight of the) connection. The probability of i (given that it is chosen) reproducing on j when i is resident (mutant) is by definition equal to some *weight* $w_{ij}^R(w_{ij}^M)$, thus $\sum_{j=1}^n w_{ij}^R = \sum_{j=1}^n w_{ij}^M = 1$ for every $i \in V$. For G_R, every edge $(ij)_R$ has weight $w_{ij}^R > 0$ if $(ij)_R \in E_R$, and $w_{ij}^R = 0$ otherwise. Similarly for G_M. For each graph we then define *weight matrices* $W_R = \left[w_{ij}^R\right]$ and $W_M = \left[w_{ij}^M\right]$ which contain all the information of the two graphs' structure. After each time-step three outcomes can occur: (i) a node is added to the *mutant set* $S \subseteq V$, (ii) a node is deleted from S, or (iii) S remains the same. If both graphs are strongly connected the process ends with probability 1 when either $S = \varnothing$ (*extinction*) or $S = V$ (*fixation*). An example is shown in Fig. 1.



$$W_R = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{bmatrix} \qquad W_M = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

Fig. 1. The 2 graphs combined; the edges of the resident graph are blue and the edges of the mutant graph are red. The respective weight matrices capture all the structure's information, including the weights to each edge. For example, the resident behaviour for node 1 (if chosen) is to reproduce only on node 2, while its mutant behaviour is to reproduce equiprobably on either 2 or 3. (Color figure online)

We denote by $f(S)$ the probability of fixation given that we start with the mutant set S. We define the *fixation probability* to be $f = \frac{1}{n}\sum_{u \in V} f(\{u\})$ for a fixed relative fitness r. We also define the *extinction probability* to be equal to $1 - f$. In the case of only one graph G (i.e. $G_R = G_M = G$), which has been the standard setting so far, the point of reference for a graph's behaviour is the fixation probability of the complete graph (called *Moran fixation probability*) $f_{\text{Moran}} = \left(1 - \frac{1}{r}\right) / \left(1 - \frac{1}{r^n}\right)$. G is an *amplifier of selection* if $f > f_{\text{Moran}}$ and $r > 1$ or $f < f_{\text{Moran}}$ and $r < 1$ because it favors advantageous mutants and discourages disadvantageous ones. G is a *suppressor of selection* if $f < f_{\text{Moran}}$ and $r > 1$ or $f > f_{\text{Moran}}$ and $r < 1$ because it discourages advantageous mutants and favors disadvantageous ones.

An undirected graph is a graph G for which $w_{ij} \in E$ if and only if $w_{ji} \in E$. An *unweighted graph* is a graph with the property that for every $i \in V$: $w_{ij} = \frac{1}{deg(i)}$ for every j with incoming edge from i, where $deg(i)$ is the outdegree of node i.

In the sequel we will abuse the term *undirected graph* to refer to an undirected unweighted graph.

In what follows we will use special names to refer to some specific graph classes. The following graphs have n vertices which we omit from the notation for simplicity.

– CL as a shorthand for the Clique or complete graph K_n.
– UST as a shorthand for the Undirected Star graph $K_{1,n-1}$.
– UCY as a shorthand for the Undirected Cycle or 2-regular graph C_n.
– CId: as a shorthand for the Circulant graph $Ci_n(1, 2, \ldots, d/2)$ for even d. Briefly this subclass of circulant graphs is defined as follows. For even degree d, the graph CId (see Fig. 2 in the full paper [11]) has vertex set $\{1, 2, \cdots, n\}$, and each vertex i is connected to vertices $\{(i - 1 \pm k) \mod n + 1 : k = 1, \ldots, d/2\}$.

By "*Resident Graph vs Mutant Graph*" we refer to the process with $G_R =$ *Resident Graph* and $G_M =$ *Mutant Graph* and by f_{G_R,G_M} we refer to the fixation probability of that process.

We note that in this paper, we are interested in the asymptotic behavior of the fixation probability in the case where the population size n is large. Therefore, we employ the standard asymptotic notation with respect to n; in particular, r is almost always treated as a variable independent of n. Furthermore, in the rest of the paper, by G_R and G_M we mean graph classes $\{(G_R)_n\}_{n\geq 3}$ and $\{(G_M)_n\}_{n\geq 3}$ respectively, and we will omit the n since we only care about the fixation probability when $n \to \infty$.

3 Our Results

In this paper, we introduce and study for the first time a generalization of the model of [10] by assuming that different types of individuals perceive the population through different graphs defined on the same vertex set, namely $G_R = (V, E_R)$ for residents and $G_M = (V, E_M)$ for mutants. In this model, we study the fixation probability, i.e. the probability that eventually only mutants remain in the population, for various pairs of graphs.

In particular, in Sect. 5 we initially prove a tight upper bound (Theorem 1) on the fixation probability for the general case of an arbitrary pair of digraphs. Next, we prove a generalization of the Isothermal Theorem of [10], that provides sufficient conditions for a pair of graphs to have fixation probability equal to the fixation probability of a clique pair, namely $f_{\text{Moran}} \overset{def}{=} f_{CL,CL} = \left(1 - \frac{1}{r}\right) / \left(1 - \frac{1}{r^n}\right)$; this corresponds to the absorption probability of a simple birth-death process with forward bias r. It is worth noting that it is easy to find small counterexamples of pairs of graphs for which at least one of the two conditions of Theorem 2 does not hold and yet the fixation probability is equal to f_{Moran}; hence we do not prove necessity.

In Sect. 6 we give a 2-player strategic game view of the process where player payoffs correspond to fixation and/or extinction probabilities. In this setting, we give an extensive study of the fixation probability when one of the two underlying graphs is complete, providing several insightful results. In particular, we prove that, the fixation probability $f_{UST,CL}$ when the mutant graph is the clique on n vertices (i.e. $G_M = CL$) and the resident graph is the undirected star on n vertices (i.e. $G_R = UST$) is $1-O(1/n)$, and thus tends to 1 as the number of vertices grows, for any constant $r > 0$. By using a translation result (Lemma 1), we can show that, when the two graphs are exchanged, then $f_{CL,UST} \to 0$. However, using a direct proof, in Theorem 4 we show that in fact $f_{CL,UST} \in O\left(\frac{r^{n-1}}{(n-2)!}\right)$, i.e. it is exponentially small in n, for any constant $r > 0$. In Theorem 6, we also provide a lower bound on the fixation probability in the special case where the resident graph is any undirected graph and the mutant graph is a clique.

Furthermore, in Subsect. 6.3, we find bounds on the fixation probability when the mutant graph is the clique and the resident graph belongs to various classes of regular graphs. In particular, we show that when the mutant graph is the clique and the resident graph is the undirected cycle, then $1 - \frac{1}{r} - o(1) \leq f_{UCY,CL} \leq \frac{1}{e^{1/r} - o(1)}$, for any constant $r > 2$. A looser lower bound holds for smaller values of r. This in particular implies that the undirected cycle is quite resistant to the clique. Then, we analyze the fixation probability by replacing the undirected cycle by 3 increasingly denser circulant graphs and find that, the denser the graph, the smaller r is required to achieve a $1 - 1/r$ asymptotic lower bound. We also find that the asymptotic upper bound stays the same when the resident graphs become denser with constant degree, but it goes to $1 - 1/r$ when the degree is $\omega(1)$. In addition, by running simulations (which we do not analyse here) for the case where the resident graph is the strongest known suppressor, i.e. the one in [5], and the mutant graph is the clique, we get fixation probability significantly greater than f_{Moran} for up to 336 nodes and values of fitness $r > 2$. All of our results seem to indicate that the clique is the most beneficial graph (in terms of player payoff in the game theoretic formulation). However, we leave this fact as an open problem for future research.

Finally, in Sect. 7 we consider the problem of efficiently approximating the fixation probability in our model. We point out that Theorem 4 implies that the fixation probability cannot be approximated via a method similar to [2]. However, when we restrict the mutant graph to be complete, we prove a polynomial (in n) upper bound for the absorption time of the generalized Moran process when $r > 2c\,(1 + o(1))$, where c is the maximum ratio of degrees of adjacent nodes in the resident graph. The latter allows us to give a fully polynomial randomized approximation scheme (FPRAS) for the problem of computing the fixation probability in this case.

Some proofs are omitted due to lack of space, and can be found in the full version of the paper [11].

4 Previous Work

So far the bibliography consists of works that consider the same structure for both residents and mutants. This 1-graph setting was initiated by Moran [14] where the case of the complete graph was examined. Many years later, the setting was extended to structured populations on general directed graphs by Lieberman et al. [10]. They introduced the notions of amplifiers and suppressors of selection, a categorization of graphs based on the comparison of their fixation probabilities with that of the complete graph. They also found a sufficient condition (in fact [4] corrects the claim in [10] that the condition is also necessary) for a digraph to have the fixation probability of the complete graph, but a necessary condition is yet to be found.

Since the generalized 1-graph model in [10] was proposed, a great number of works have tried to answer some very intriguing questions in this framework. One of them is the following: which are the best unweighted amplifiers and suppressors that exist? Díaz et al. [2] give the following bounds on the fixation probability of strongly connected digraphs: an upper bound of $1-\frac{1}{r+n}$ for $r>0$, a lower bound of $\frac{1}{n}$ for $r>1$ and they show that there is no positive polynomial lower bound when $0<r<1$. An interesting problem that was set in [10] is whether there are graph families that are *strong amplifiers* or *strong suppressors* of selection, i.e. families of graphs with fixation probability tending to 1 or to 0 respectively as the order of the graph tends to infinity and for $r>1$. Galanis et al. [4] find an infinite family of strongly-amplifying directed graphs, namely the "megastar" with fixation probability $1-O(n^{-1/2}\log^{23} n)$, which was later proved to be optimal up to logarithmic factors [6].

While the search for optimal directed strong amplifiers was still on, a restricted version of the problem had been drawing a lot of attention: which are the tight bounds on the fixation probability of undirected graphs? The lower bound in the undirected case remained $\frac{1}{n}$, but the upper bound was significantly improved by Mertzios and Spirakis [13] to $1-\Omega(n^{-3/4})$, when r is independent of n. It was again improved by Giakkoupis [5] to $1-\Omega\left(\frac{1}{\epsilon}n^{-1/3}\log n\right)$ for $r\geq 1+\epsilon$ where $0<\epsilon\leq 1$, and finally by Goldberg et al. [6] to $1-\Omega(n^{-1/3})$ where they also find a graph which shows that this is tight. While the general belief was that there are no undirected strong suppressors, Giakkoupis [5] showed that there is a class of graphs with fixation probability $O(r^2 n^{-1/4}\log n)$, opening the way for a potentially optimal strong suppressor to be discovered.

Extensions of [10] where the interaction between individuals includes a bimatrix game have also been studied. Ohtsuki et al. in [16] considered the generalized Moran process with two distinct graphs, where one of them determines possible pairs that will play a bimatrix game and yield a total payoff for each individual, and the other determines which individual will be replaced by the process in each step. Two similar settings, where a bimatrix game determines the individuals' fitness, were studied by Ibsen-Jensen et al. in [8]. In that work they prove NP-completeness and #P-completeness on the computation of the fixation probabilities for each setting.

5 Markov Chain Abstraction and the Generalized Isothermal Theorem

This generalized process with two graphs we propose can be modelled as an absorbing Markov chain [15]. The states of the chain are the possible mutant sets $S \subseteq V$ (2^n different mutant sets) and there are two absorbing states, namely $\langle \varnothing \rangle$ and $\langle V \rangle$. In this setting, the fixation probability is the average absorption probability to $\langle V \rangle$, starting from a state with one mutant. Since our Markov chain contains only two absorbing states, the sum of the fixation and extinction probabilities is equal to 1.

Transition probabilities. In the sequel we will denote by $X+y$ the set $X \cup \{y\}$ and by $X - y$ the set $X \setminus \{y\}$. We can easily deduce the boundary conditions from the definition: $f(\varnothing) = 0$ and $f(V) = 1$. For any other arbitrary state $\langle S \rangle$ of the process we have:

$$\begin{aligned} f(S) = & \sum_{i \in S, j \notin S} \frac{r}{F(S)} w_{ij}^M \cdot f(S+j) + \sum_{j \notin S, i \in S} \frac{1}{F(S)} w_{ji}^R \cdot f(S-i) \\ & + \left(\sum_{i \in S, j \in S} \frac{r}{F(S)} w_{ij}^M + \sum_{i \notin S, j \notin S} \frac{1}{F(S)} w_{ij}^R \right) \cdot f(S), \end{aligned} \tag{1}$$

where $F(S) = |S|r + |V| - |S|$ is the total fitness of the population in state $\langle S \rangle$. By eliminating self-loops, we get

$$f(S) = \frac{\sum_{i \in S, j \notin S} r \cdot w_{ij}^M \cdot f(S+j) + \sum_{j \notin S, i \in S} w_{ji}^R \cdot f(S-i)}{\sum_{i \in S, j \notin S} r \cdot w_{ij}^M + \sum_{j \notin S, i \in S} w_{ji}^R}. \tag{2}$$

We should note here that, in the general case, the fixation probability can be computed by solving a system of 2^n linear equations using this latter relation. However, bounds are usually easier to be found and special cases of resident and mutant graphs may have efficient exact solutions.

Using the above Markov chain abstraction and stochastic domination arguments we can prove the following general upper bound on the fixation probability:

Theorem 1. *For any pair of digraphs G_R and G_M with $n = |V|$, the fixation probability f_{G_R,G_M} is upper bounded by $1 - \frac{1}{r+n}$, for $r > 0$. This bound is tight for r independent of n.*

We now prove a generalization of the Isothermal Theorem of [10].

Theorem 2 (Generalized Isothermal Theorem). *Let $G_R(V, E_R)$, $G_M(V, E_M)$ be two directed graphs with vertex set V and edge sets E_R and E_M respectively. The generalized Moran process with 2 graphs as described above has the Moran fixation probability if:*

1. *$\sum_{j \neq i} w^R_{ji} = \sum_{j \neq i} w^M_{ji} = 1$, $\forall i \in V$, that is, W_R and W_M are doubly stochastic, i.e. G_R and G_M are isothermal (actually one of them being isothermal is redundant as it follows from the second condition), and*
2. *for every pair of nodes $i, j \in V$: $w^R_{ij} + w^R_{ji} = w^M_{ij} + w^M_{ji}$.*

Observe that when $G_R = G_M$ we have the isothermal theorem of the special case of the generalized Moran process that has been studied so far.

6 A Strategic Game View

In this section we study the aforementioned process from a game-theoretic point of view. Consider the strategic game with 2 players; residents (type R) and mutants (type M), so the player set is $N = \{R, M\}$. The action set of a player $k \in N$ consists of all possible strongly connected graphs[1] $G_k(V, E_k)$ that she can construct with the available vertex set V. The payoff for the residents (player R) is the probability of extinction, and the payoff for the mutants (player M) is the probability of fixation. Of course, the sum of payoffs equals 1, so the game can be reduced to a zero-sum game.

The natural question that emerges is: what are the pure Nash equilibria of this game (if any)? For example, for fixed $r > 1$, if we only consider two actions for every player, namely the graphs CL and UST, then from our results from Subsect. 6.1, when $n \to \infty$, we get $f_{CL,UST} \to 0, f_{UST,CL} \to 1$ and from [1,15], $f_{CL,CL} \to 1 - 1/r$ and $f_{UST,UST} \to 1 - 1/r^2$. Therefore, we get the following bimatrix game:

		Player M	
		CL	UST
Player R	CL	$1/r, 1 - 1/r$	$1, 0$
	UST	$0, 1$	$1/r^2, 1 - 1/r^2$

which has a pure Nash equilibrium, namely (CL, CL). Trying to understand better the behaviour of the two conflicting graphs, we put some pairs of them to the test. The main question we ask in this work is: what is the best response graph G_R of the residents to the Clique graph of the mutants? In the sequel, we will use the abbreviations *pl-R* and *pl-M* for the resident and the mutant population, respectively.

6.1 Star vs Clique

The following result implies (since $(n-4)!^{-1/(n-2)} \to 0$ as $n \to \infty$) that when the mutant graph is complete and the resident graph is the undirected star, the fixation probability tends to 1 as n goes to infinity.

[1] We assume strong connectivity in order to avoid problematic cases where there is neither fixation nor extinction.

Theorem 3. *If pl-R has the UST graph and pl-M has the CL graph for $r > (n-4)!^{-1/(n-2)}$, then the payoff of pl-M (fixation probability) is lower bounded by $\frac{1-\frac{1}{n}}{1+\frac{1}{r(n-2)}+\frac{1}{r^2(n-3)}} > 1 - \frac{1}{n} - \frac{1}{r(n-2)} - \frac{1}{r^2(n-3)}$.*

It is worth noting that, since the game we defined in Subsect. 6 is 1-sum, we immediately can get upper (resp. lower) bounds on the payoff of pl-R, given lower (resp. upper) bounds on the payoff of pl-M.

Now we give the following lemma that connects the fixation probability of a process with given relative fitness, resident and mutant graphs, with the fixation probability of a "mirror" process where the roles between residents and mutants are exchanged.

Lemma 1. $f_{G_R,G_M}(r) \leq 1 - f_{G_M,G_R}(\frac{1}{r})$.

This result provides easily an upper bound on the fixation probability of a given process when a lower bound on the fixation probability is known for its "mirror" process. For example, using Theorem 3 and Lemma 1 we get an upper bound $\frac{1}{n} + \frac{1}{r(n-2)} + \frac{1}{r^2(n-3)}$ for $r > 0$ on the fixation probability of CL vs UST; this immediately implies that the probability of fixation in this case tends to 0. However, as we subsequently explain, a more precise lower bound is necessary to reveal the approximation restrictions of the particular process.

Theorem 4. *If pl-R has the CL graph and pl-M has the UST graph for $r > 0$, then the payoff of pl-M (fixation probability) is upper bounded by $\frac{r^{n-1}}{(n-2)!}$.*

This bound shows that, not only there exists a graph that suppresses selection against the UST (which is an amplifier in the 1-graph setting), but it also does that with great success. In fact for any mutant with constant r arbitrarily large, its fixation probability is less than exponentially small.

In view of the above, the following result implies that the fixation probability in our model cannot be approximated via a method similar to [2].

Theorem 5 (Bounds on the 2-graphs Moran process). *There is a pair of graphs G_R, G_M such that the fixation probability f_{G_R,G_M} is $o\left(\frac{1}{a^n}\right)$, for some constant $a > 1$, when the relative fitness r is constant. Furthermore, there is a pair of graphs G'_R, G'_M such that the fixation probability $f_{G'_R,G'_M}$ is at least $1 - O\left(\frac{1}{n}\right)$, for constant $r > 0$.*

6.2 Arbitrary Undirected Graphs vs Clique

The following result is a lower bound on the fixation probability.

Theorem 6. *When pl-R has an undirected graph for which $w^R_{xy}/w^R_{yx} \leq c$ for every $(xy) \in E_R$ and pl-M has the CL graph, the payoff of pl-M (fixation probability) is lower bounded by $\left[\frac{1-\left(\frac{c}{r}\right)^{\log n}}{1-\frac{c}{r}}(1+o(1)) + \frac{\left(\frac{2c}{r}\right)^{\log n} - \left(\frac{2c}{r}\right)^n}{1-\frac{2c}{r}}\right]^{-1}$, for $r > 0$. In particular, for $r > 2c$ the lower bound tends to $1 - \frac{c}{r}$ as $n \to \infty$.*

Proof. Notice that, given the number of mutants at a time-step is $i := |S|$, the probability that a resident becomes mutant is $p_i^{i+1} = \frac{ir}{ir+n-i} \cdot \frac{n-i}{n-1}$, and the probability that a mutant becomes resident p_i^{i-1} is upper bounded by $\frac{\min\{i,n-i\}}{ir+n-i} \max\limits_{(xy)\in E_R} \frac{w_{xy}^R}{w_{yx}^R}$. That is because the maximum possible number of resident-to-mutant edges in G_R at a step with i mutants is achieved when either every mutant has edges in G_R only towards residents, or every resident has edges in G_R only towards mutants; and the most extreme case is when every one of the $\min\{i, n-i\}$ nodes has sum of weights of incoming edges equal to the maximum ratio of degrees of adjacent nodes in G_R, i.e. $c := \max\limits_{(xy)\in E_R} \frac{w_{xy}^R}{w_{yx}^R}$.

This means that the number of mutants in our given process P of an undirected graph vs Clique stochastically dominates a birth-death process P' that is described by the following Markov chain: A state $\langle i\rangle$, where $i \in \{0, 1, 2, \ldots, n\}$ is the number of mutants on the vertex set and the only absorbing states are $\langle 0\rangle$ and $\langle n\rangle$. At this point we state the following fact from [15]:

Fact 1. *In a birth-death process with state space* $\{0, 1, \ldots, n\}$*, absorbing states* $0, n$ *and backward bias at state* k *equal to* γ_k*, the probability of absorption at* n*, given that we start at* i *is* $f_i = (1 + \sum_{j=1}^{i-1} \prod_{k=1}^{j} \gamma_k)/(1 + \sum_{j=1}^{n-1} \prod_{k=1}^{j} \gamma_k)$.

Using Fact 1 in our process we get: $f_1 = 1/\left(1 + \sum_{j=1}^{n-1} \prod_{k=1}^{j} \gamma_k\right)$, where $\gamma_i = p_i^{i-1}/p_i^{i+1}$. From the aforementioned transition probabilities of our Markov chain we have:

$$\gamma_k \leq \begin{cases} \frac{c}{r} \cdot \frac{n-1}{n-k}, & \text{for } k \in \{1, 2, \ldots, \lfloor \frac{n}{2} \rfloor\} \\ \frac{c}{r} \cdot \frac{n-1}{k}, & \text{for } k \in \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n-1\} \end{cases}$$

Now we can calculate a lower bound on the fixation probability of P' using the fact that $\frac{n-1}{n-2} = 1 + \frac{1}{n-2}, \frac{n-1}{n-3} = 1 + \frac{2}{n-3}, \cdots, \frac{n-1}{n-\log n+1} = 1 + \frac{\log n-2}{n-\log n+1}$:

$$f_1 = \frac{1}{\left[\sum_{j=0}^{\log n-1} \left(\frac{c}{r}\right)^j\right](1+o(1)) + \frac{\left(\frac{c}{r}\right)^{\log n}(n-1)^{\log n}}{(n-1)\cdots(n-\log n)} + \cdots + \frac{\left(\frac{c}{r}\right)^{n-1}(n-1)^{n-1}}{\left[(n-1)\cdot\cdots\cdot\left(\frac{n}{2}+1\right)\right]^2\cdot\left(\frac{n}{2}\right)}}$$

$$\geq \frac{1}{\frac{1-\left(\frac{c}{r}\right)^{\log n}}{1-\frac{c}{r}}(1+o(1)) + \left(\frac{2c}{r}\right)^{\log n} \sum_{j=0}^{n-\log n-1} \left(\frac{2c}{r}\right)^j} \quad , \quad (\text{since } \gamma_k \leq \frac{2c}{r})$$

$$= \frac{1}{\frac{1-\left(\frac{c}{r}\right)^{\log n}}{1-\frac{c}{r}}(1+o(1)) + \left(\frac{2c}{r}\right)^{\log n} \frac{1-\left(\frac{2c}{r}\right)^{n-\log n}}{1-\frac{2c}{r}}}.$$

$\square$

From the theorem above it follows that if G_R is undirected regular then the fixation probability of G_R vs CL is lower bounded by $1 - 1/r$ for $r > 2$ and $n \to \infty$, which equals f_{Moran} (defined in Sect. 2).

We also note that, by Lemma 1 and the above theorem, when $G_R = CL$, G_M is an undirected graph with $w_{xy}^M/w_{yx}^M \leq c$ for every $(xy) \in E_M$, and relative fitness $r < \frac{1}{2c}$, then the upper bound of the fixation probability tends to cr as $n \to \infty$.

6.3 Circulant Graphs vs Clique

In this subsection we give bounds for the fixation probability of CId vs CL. We first prove the following result that gives an upper bound on the fixation probability when G_R is the CId graph as described in Sect. 2 and G_M is the complete graph on n vertices.

Theorem 7. *When mutants have the CL graph, if residents have a CId graph and $d \in \Theta(1)$, then the payoff of pl-M (fixation probability) is upper bounded by $\left[e^{\frac{1}{r}} - \frac{1}{r^n}\frac{1}{n!}\frac{1}{1-\frac{1}{r}}\right]^{-1}$ for $r > 1$ and $\left[e^{\frac{1}{r}} - \frac{1}{r^n}\frac{1}{n!} - o(1)\right]^{-1}$ for $r \leq 1$. In particular, for constant $r > 0$ the upper bound tends to $e^{-\frac{1}{r}}$. If $d \in \omega(1)$, then the upper bound is $\left(1 - \frac{1}{r}\right)\left[1 - \frac{1}{r^{g(n)}} - o(1)\right]^{-1}$, for $r > 0$, where $g(n)$ is a function of n such that $g(n) \in \omega(1)$ and $g(n) \in o(d)$. The bound improves as $g(n)$ is picked closer to $\Theta(d)$ and, in particular, for $r > 1$ it tends to $1 - \frac{1}{r}$.*

We also show that our upper bound becomes tighter as d increases. In particular, we prove the following lower bounds:

Theorem 8. *When mutants have the CL graph, if residents have the UCY (degree $d=2$) or a graph of the class CId for degree $d=4$, 6 or 8, then the payoff of pl-M (fixation probability) is lower bounded by $\left[\frac{1-\left(\frac{1}{r}\right)^{\log n}}{1-\frac{1}{r}}(1+o(1)) + \frac{\left(\frac{c}{r}\right)^{\log n} - \left(\frac{c}{r}\right)^n}{1-\frac{c}{r}}\right]^{-1}$, where $c = \frac{d+2}{d}$ for $r > 0$. In particular, for $r > c$ the lower bound tends to $1 - \frac{1}{r}$ as $n \to \infty$.*

By the above two theorems, we get the following:

Corollary 1. *If $G_R = UCY$ (or G_R is one of $CI4$, $CI6$ and $CI8$), $G_M = CL$, and $r > 2$ (respectively $r > \frac{3}{2}$, $r > \frac{4}{3}$ and $r > \frac{5}{4}$), then f_{G_R,G_M} tends to a positive constant smaller than 1 as $n \to \infty$.*

Finally, we note that, by Lemma 1 and the above Corollary, when the resident graph is complete (i.e. $G_R = CL$), the mutant graph is UCY (or one of $CI4$, $CI6$, $CI8$), and the relative fitness satisfies $r < \frac{1}{2}$ (respectively $r < \frac{2}{3}$, $r < \frac{3}{4}$ and $r < \frac{4}{5}$), then the fixation probability is upper bounded by a constant smaller than 1, as $n \to \infty$.

7 An Approximation Algorithm

Here we present a fully polynomial randomized approximation scheme (FPRAS)[2] for the problem UndirectedVsClique of computing the fixation probability in the Moran process when the residents have an undirected graph and the

[2] An FPRAS for a function f that maps problem instances to numbers is a randomized algorithm with input X and parameter $\epsilon > 0$, which is polynomial in $|X|$ and ϵ^{-1} and outputs a random variable g, such that $Pr\{(1-\epsilon)f(X) \leq g(X) \leq (1+\epsilon)f(X)\} \geq \frac{3}{4}$ [9].

mutants have the clique graph with $r > 2c\left(1 + \frac{2}{n-5}\right)$, where c is the maximum ratio of the degrees of adjacent nodes in the resident graph. The following result is essential for the design of a FPRAS; it gives an upper bound (which depends on c, r and is polynomial in n) on the expected absorption time of the Moran process in this case.

Theorem 9. *Let $G_R(V, E_R)$ be an undirected graph of order n, for which $w^R_{xy}/w^R_{yx} \leq c$ for every $(xy) \in E_R$. Let $G_M(V, E_M)$ be the clique graph of order n. For $r \geq 2c\left(1 + \frac{2}{n-5}\right)$ and any $S \subseteq V$, the absorption time τ of the Moran process "G_R vs G_M" satisfies:*

$$\mathbb{E}[\tau | X_0 = S] \leq \frac{r}{r-c} n(n - |S|).$$

In particular, $\mathbb{E}[\tau] \leq \frac{r}{r-c} n^2$.

For our algorithm to run in time polynomial in the length of the input, r must be encoded in unary.

Theorem 10. *There is an FPRAS for* UndirectedVsClique*, for $r > 2c\left(1 + \frac{2}{n-5}\right)$.*

Proof. We present the following algorithm. First, we find the constant c by checking every edge of the resident graph and exhaustively finding the maximum ratio of adjacent nodes' degrees in $O(n^3)$ time. If and only if our r is greater than $2c\left(1 + \frac{2}{n-5}\right)$, we simulate the Moran process where residents have some given undirected graph and mutants have the clique graph. We compute the proportion of simulations that reached fixation for $N = \lceil 2\epsilon^{-2} \ln 16 \rceil$ simulation runs with maximum number $T = \lceil 8rn^2N(r-c)^{-1} \rceil$ of steps each. In case of simulations that do not reach absorption in the T-th step, the simulation stops and returns an error value.

Also, each transition of the Moran process can be simulated in $O(1)$ time. This is possible if we keep track of the resident and mutant nodes in an array, thus choose the reproducing node in constant time. Further, we can pick the offspring node in constant time by running a breadth-first search for each graph before the simulations start, storing the neighbours of each node for the possible node types (resident and mutant) in arrays. Hence the total running time is $O(n^3 + NT)$, which is polynomial in n and ϵ^{-1} as required by the FPRAS definition.

Now, we only have to show that the output of our algorithm computes the fixation probability to within a factor of $1 \pm \epsilon$ with probability at least 3/4. Essentially, the proof is the same as in [2] with modifications needed for our setting. For $i \in \{1, 2, \ldots, N\}$, let Y_i be the indicator variable, where $Y_i = 1$ if the i-th simulation of the Moran process reaches fixation and $Y_i = 0$ otherwise. We first calculate the bounds on the probability of producing an output of error ϵ in the event where all simulation runs reach absorption within T steps. The output of our algorithm is then $g = \frac{1}{N}\sum_{i=1}^{N} Y_i$ while the required function is the fixation probability f. Using Hoeffding's inequality we get:

$$Pr\{|g - f| > \epsilon f\} \leq 2e^{-2\epsilon^2 f^2 N} \leq 2e^{-f^2 \ln 16/4} < \frac{1}{8}$$

where the latter inequality is because $f \geq 1 - c/r > 1/2$ due to Theorem 6.

Now, by using Theorem 9 and Markov's inequality, the process reaches absorption within t steps with probability at least $1-\epsilon$, for any $\epsilon \in (0, 1)$ and any $t \geq \frac{r}{r-c} n^2 \frac{1}{\epsilon}$. Therefore, the event that any individual simulation has not reached absorption within T steps, happens with probability at most $1/(8N)$. By taking the union bound, the event of a simulation run not reaching absorption within T steps happens with probability at most 1/8. Thus, the probability of producing an output g as required, is at least 3/4. □

References

1. Broom, M., Rychtář, J.: An analysis of the fixation probability of a mutant on special classes of non-directed graphs. Proc. Roy. Soc. Lond. A: Math. Phys. Eng. Sci. **464**(2098), 2609–2627 (2008)
2. Díaz, J., Goldberg, L.A., Mertzios, G.B., Richerby, D., Serna, M., Spirakis, P.G.: Approximating fixation probabilities in the generalized Moran process. Algorithmica **69**(1), 78–91 (2014)
3. Easley, D., Kleinberg, J.: Networks, Crowds, and Markets, vol. 6, no. 1, pp. 1–6. Cambridge University Press, Cambridge (2010)
4. Galanis, A., Göbel, A., Goldberg, L.A., Lapinskas, J., Richerby, D.: Amplifiers for the Moran process. In: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, 11–15 July 2016, Rome, Italy, pp. 62:1–62:13 (2016)
5. Giakkoupis, G.: Amplifiers and suppressors of selection for the Moran process on undirected graphs. CoRR, abs/1611.01585 (2016)
6. Goldberg, L.A., Lapinskas, J., Lengler, J., Meier, F., Panagiotou, K., Pfister, P.: Asymptotically optimal amplifiers for the Moran process. ArXiv e-prints, November 2016
7. Hofbauer, J., Sigmund, K.: Evolutionary Games and Population Dynamics. Cambridge University Press, Cambridge (1998)
8. Ibsen-Jensen, R., Chatterjee, K., Nowak, M.A.: Computational complexity of ecological and evolutionary spatial dynamics. Proc. Nat. Acad. Sci. **112**(51), 15636–15641 (2015)
9. Karp, R.M., Luby, M.: Monte-Carlo algorithms for enumeration and reliability problems. In: 24th Annual Symposium on Foundations of Computer Science, pp. 56–64. IEEE (1983)
10. Lieberman, E., Hauert, C., Nowak, M.A.: Evolutionary dynamics on graphs. Nature **433**(7023), 312–316 (2005)
11. Melissourgos, T., Nikoletseas, S.E., Raptopoulos, C., Spirakis, P.G.: Mutants and residents with different connection graphs in the Moran process. CoRR, abs/1710.07365 (2017)

12. Mertzios, G.B., Nikoletseas, S., Raptopoulos, C., Spirakis, P.G.: Natural models for evolution on networks. In: Chen, N., Elkind, E., Koutsoupias, E. (eds.) WINE 2011. LNCS, vol. 7090, pp. 290–301. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25510-6_25
13. Mertzios, G.B., Spirakis, P.G.: Strong bounds for evolution in networks. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 669–680. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_58
14. Moran, P.A.P.: Random processes in genetics. Math. Proc. Camb. Philos. Soc. **54**(1), 6071 (1958)
15. Nowak, M.A.: Evolutionary Dynamics: Exploring the Equations of Life. Harvard University Press, Cambridge (2006)
16. Ohtsuki, H., Pacheco, J.M., Nowak, M.A.: Evolutionary graph theory: breaking the symmetry between interaction and replacement. J. Theor. Biol. **246**(4), 681–694 (2007)

A Framework for Algorithm Stability and Its Application to Kinetic Euclidean MSTs

Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms(✉)

Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands
{w.meulemans,b.speckmann,k.a.b.verbeek,j.j.h.m.wulms}@tue.nl

Abstract. We say that an algorithm is *stable* if small changes in the input result in small changes in the output. This kind of algorithm stability is particularly relevant when analyzing and visualizing time-varying data. Stability in general plays an important role in a wide variety of areas, such as numerical analysis, machine learning, and topology, but is poorly understood in the context of (combinatorial) algorithms.

In this paper we present a framework for analyzing the stability of algorithms. We focus in particular on the tradeoff between the stability of an algorithm and the quality of the solution it computes. Our framework allows for three types of stability analysis with increasing degrees of complexity: event stability, topological stability, and Lipschitz stability. We demonstrate the use of our stability framework by applying it to kinetic Euclidean minimum spanning trees.

1 Introduction

With recent advances in sensing technology, vast amounts of *time-varying data* are generated, processed, and analyzed on a daily basis. Hence there is a great need for algorithms that can operate efficiently on time-varying data and that can offer guarantees on the quality of analysis results. A specific relevant subset of time-varying data consists of so-called *motion data*: geolocated, and hence geometric, time-varying data. To deal with the challenges of motion data, Basch *et al.* [3] in 1999 introduced the *kinetic data structures* (KDS) framework. Kinetic data structures efficiently maintain a (combinatorial) structure on a set of moving objects. The KDS framework has sparked a significant amount of research, resulting in many efficient algorithms for motion data.

The performance of a particular algorithm is usually judged with respect to a variety of criteria, with the two most common being solution quality and

W. Meulemans and J. Wulms are (partially) supported by the Netherlands eScience Center (NLeSC) under grant number 027.015.G02. B. Speckmann and K. Verbeek are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208 and no. 639.021.541, respectively.

running time. In the context of algorithms for time-varying data, a third important criterion is *stability*. Whenever analysis results need to be communicated to humans, for example via visual representations, it is important that these results are *stable*: small changes in the data result in small changes in the output. These changes in the output are continuous or discrete depending on the algorithm: graphs usually undergo discrete changes while a convex hull of moving points changes continuously. Sudden changes in the visual representation of data disrupt the so-called *mental map* [16] of the user and prevent the recognition of temporal patterns. Stability also plays a role if changing the result is costly in practice (e.g. in physical network design), where frequent significant changes to the network are prohibitively expensive.

The stability of algorithms or methods has been well-studied in a variety of research areas, such as numerical analysis [14], machine learning [5], control systems [2], and topology [7]. In contrast, the stability of combinatorial algorithms for time-varying data has received little attention in the theoretical computer science community so far. Here it is of particular interest to understand the tradeoffs between solution quality, running time, and stability. As an example, consider maintaining a minimum spanning tree of a set of moving points. If the points move, it might have to frequently change significantly. On the other hand, if we start with an MST for the input point set and then never change it combinatorially as the points move, the spanning tree we maintain is very stable – but over time it can devolve to a low quality and very long spanning tree.

Our goal, and the focus of this paper, is to understand the possible tradeoffs between solution quality and stability. This is in contrast to earlier work on stability in other research areas, such as the ones mentioned above, where stability is usually considered in isolation. Since there are currently no suitable tools available to formally analyze tradeoffs involving stability, we introduce a new analysis framework. We believe that there are many interesting and relevant questions to be solved in the general area of algorithmic stability analysis and we hope that our framework is a first meaningful step towards tackling them.

Results and organization. We present a framework to analyze the stability of (combinatorial) algorithms. As a first step, we limit ourselves to analyzing the tradeoff between stability and solution quality, omitting running time from consideration. Our framework allows for three types of stability analysis of increasing degrees of complexity: *event stability*, *topological stability*, and *Lipschitz stability*. It can be applied both to motion data and to more general time-varying data. We demonstrate the use of our stability framework by applying it to the problem of kinetic Euclidean minimum spanning trees (EMSTs). Some of our results for kinetic EMSTs are directly more widely applicable.

In Sect. 2 we give an overview of our framework for algorithm stability. In Sects. 3, 4, and 5 we describe event stability, topological stability, and Lipschitz stability, respectively. In each of these sections we first describe the respective type of stability analysis in a generic setting, followed by specific results using that type of stability analysis on the kinetic EMST problem. In Sect. 6 we make some concluding remarks on our stability framework. Omitted proofs can be found in the full version of the paper.

Related work. Stability is a natural point of concern in more visual and applied research areas such as graph drawing, (geo-)visualization, and automated cartography. For example, in dynamic map labelling [4], the *consistent dynamic labelling* model allows a label to appear and disappear only once, making it very stable. There are very few theoretical results, with the noteworthy exception of so-called simultaneous embeddings [6] in graph drawing, which can be seen as a very restricted model of stability. However, none of these results offer any real structural insight into the tradeoff between solution quality and stability.

In computational geometry there are a few results on the tradeoff between solution quality and stability. Specifically, Durocher and Kirkpatrick [9] study the stability of centers of kinetic point sets, and define the notion of κ-stable center functions, which is closely related to our concept of Lipschitz stability. In later work [10] they consider the tradeoff between the solution quality of Euclidean 2-centers and a bound on the velocity with which they can move. De Berg *et al.* [8] show similar results in the black-box KDS model. One can argue that the KDS framework [13] already indirectly considers stability in a limited form, namely as the number of *external events*. However, the goal of a KDS is typically to reduce the running time of the algorithm, and rarely to sacrifice the running time or solution quality to reduce the number of external events.

Kinetic Euclidean minimum spanning trees have been studied extensively. Katoh *et al.* [15] proved an upper bound of $O(n^3 2^{\alpha(n)})$ for the number of external events of EMSTs of n linearly moving points, where $\alpha(n)$ is the inverse Ackermann function. The best known lower bound for external events of EMSTs in d dimensions is $\Omega(n^d)$ [18]. There also exists a lot of related work on approximations of EMSTs and related structures like Delaunay triangulations, but the number of external events still remains at least roughly $\Omega(n^2)$. Our stability framework allows us to reduce the number of external events even further and to still state something meaningful about the quality of the resulting EMSTs.

2 Stability Framework

Intuitively, we can say that an algorithm is stable if small changes in the input lead to small changes in the output. More formally, let Π be an optimization problem that, given an input instance I from a set $\mathcal{I}$, asks for a feasible solution S from a set $\mathcal{S}$ that minimizes (or maximizes) some optimization function $f \colon \mathcal{I} \times \mathcal{S} \to \mathbb{R}$. An algorithm $\mathcal{A}$ for Π can be seen as a function $\mathcal{A} \colon \mathcal{I} \to \mathcal{S}$. Similarly, the optimal solutions for Π can be described by a function $\mathrm{OPT} \colon \mathcal{I} \to \mathcal{S}$. To define the stability of an algorithm, we need to quantify changes in the input instances and in the solutions. We can do so by imposing a metric on $\mathcal{I}$ and $\mathcal{S}$. Let $d_{\mathcal{I}} \colon \mathcal{I} \times \mathcal{I} \to \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{I}$ and let $d_{\mathcal{S}} \colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{S}$. We can then define the *stability* of an algorithm $\mathcal{A} \colon \mathcal{I} \to \mathcal{S}$ as follows.

$$\mathrm{St}(\mathcal{A}) = \max_{I, I' \in \mathcal{I}} \frac{d_{\mathcal{S}}(\mathcal{A}(I), \mathcal{A}(I'))}{d_{\mathcal{I}}(I, I')} \tag{1}$$

This definition for stability is closely related to that of the multiplicative distortion of metric embeddings, where $\mathcal{A}$ induces a metric embedding from the metric space $(\mathcal{I}, d_{\mathcal{I}})$ into $(\mathcal{S}, d_{\mathcal{S}})$. The lower the value for $\mathrm{St}(\mathcal{A})$, the more stable we consider the algorithm $\mathcal{A}$ to be. There are many other ways to define the stability of an algorithm given the metrics, but the above definition suffices for our purpose.

For many optimization problems, the function OPT may be very unstable. This suggests an interesting tradeoff between the stability of an algorithm and the solution quality. Unfortunately, the generic formulation of stability provided above is very unwieldy. It is not always clear how to define metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ such that meaningful results can be derived. Additionally, it is not obvious how to deal with optimization problems with continuous input and discrete solutions, where the algorithm is inherently discontinuous, and thus the stability is unbounded by definition. Finally, analyses of this form are often very complex, and it is not straightforward to formulate a simplified version of the problem. In our framework we hence distinguish three types of stability analysis: event stability, topological stability, and Lipschitz stability.

Event stability follows the setting of kinetic data structures (KDS). That is, the input (a set of moving objects) changes continuously as a function over time. However, contrary to typical KDSs where a constraint is imposed on the solution quality, we aim to enforce the stability of the algorithm. For event stability we simply disallow the algorithm to change the solution too rapidly. Doing so directly is problematic, but we formalize this approach using the concept of k-optimal solutions. As a result, we can obtain a tradeoff between stability and quality that can be tuned by the parameter k. Note that event stability captures only *how often* the solution changes, but not *how much* the solution changes at each event.

Topological stability takes a first step towards the generic setup described above. However, instead of measuring the amount of change in the solution using a metric, we merely require the solution to behave continuously. To do so we only need to define a topology on the solution space $\mathcal{S}$ that captures stable behavior. Surprisingly, even though we ignore the amount of change in a single time step, this type of analysis still provides meaningful information on the tradeoff between solution quality and stability. In fact, the resulting tradeoff can be seen as a lower bound for any analysis involving metrics that follow the used topology.

Lipschitz stability finally captures the generic setup described above. As the name suggests, we require the algorithm to be Lipschitz continuous and we provide an upper bound on the Lipschitz constant, which is equivalent to $\mathrm{St}(\mathcal{A})$. We are then again interested in the quality of the solutions that can be obtained with any Lipschitz stable algorithm. Given the complexity of this type of analysis, a complete tradeoff for any value of the Lipschitz constant is typically out of reach, but results for sufficiently small or large values can be of interest.

Remark. Our framework makes the assumption that an algorithm is a function $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$. However, in a kinetic setting this is not necessarily true, since the algorithm has *history*. More precisely, for some input instance I, a kinetic algorithm may produce different solutions for I based on the instances processed earlier. We generally allow this behavior, and for event stability this behavior is even crucial. However, for the sake of simplicity, we will treat an algorithm as a function. We also generally assume in our analysis that the input is time-varying, that is, the input is a function over time, or follows a trajectory through the input space $\mathcal{I}$. Again, for the sake of simplicity, this is not always directly reflected in our definitions. Beyond that, we operate in the black-box model, in the sense that the algorithm does not know anything about future instances.

While these are the conditions under which we use our framework, it can be applied in a variety of algorithmic settings, such as streaming algorithms and algorithms with dynamic input.

3 Event Stability

The simplest form of stability is event stability. Like the number of external events in KDSs, event stability captures only how often the solution changes.

3.1 Event Stability Analysis

Let Π be an optimization problem with a set of input instances $\mathcal{I}$, a set of solutions $\mathcal{S}$, and optimization function $f\colon \mathcal{I} \times \mathcal{S} \to \mathbb{R}$. Following the framework of kinetic data structures, we assume that the input instances include certain parameters that can change as a function of time. To apply the event stability analysis, we require that all solutions have a combinatorial description, that is, the solution description does not use the time-varying parameters of the input instance. We further require that every solution $S \in \mathcal{S}$ is feasible for every input instance $I \in \mathcal{I}$. This automatically disallows any insertions or deletions of elements. Note that an insertion or a deletion would typically force an event, and thus including this aspect in our stability analysis does not seem useful.

For example, in the setting of kinetic EMSTs, the input instances would consist of a fixed set of points. The coordinates of these points can then change as a function over time. A solution of the kinetic EMST problem consists of the combinatorial description of a tree graph on the set of input points. Note that every tree graph describes a feasible solution for any input instance, if we do not insist on any additional restrictions like, e.g., planarity. The minimization function f then simply measures the total length of the tree, for which we do need to use the time-varying parameters of the problem instance.

Rather than directly restricting the quality of the solutions, we aim to restrict the stability of any algorithm. To that end, we introduce the concept of k-optimal solutions. Let $d_{\mathcal{I}}$ be a metric on the input instances, and let $\mathrm{OPT}\colon \mathcal{I} \to \mathcal{S}$ describe the optimal solutions. We say that a solution $S \in \mathcal{S}$ is *k-optimal* for an instance $I \in \mathcal{I}$ if there exists an input instance $I' \in \mathcal{I}$ such that $f(I', S) =$

$f(I', OPT(I'))$ and $d_{\mathcal{I}}(I, I') \leq k$. With this definition any optimal solution is always 0-optimal. Note that this definition requires a form of normalization on the metric $d_{\mathcal{I}}$, similar to that of e.g. smoothed analysis [19]. We therefore require that there exists a constant c such that every solution $S \in \mathcal{S}$ is c-optimal for every instance $I \in \mathcal{I}$. For technical reasons we require the latter condition to hold only for some time interval $[0, T]$ of interest. Note that the concept of k-optimal solutions is closely related to *backward error analysis* in numerical analysis.

Following the framework of kinetic data structures, we typically require the functions of the time-varying parameters to be well-behaved (e.g., polynomial functions), for otherwise we cannot derive meaningful bounds. The event stability analysis then considers two aspects. First, we analyze how often the solution needs to change to maintain a k-optimal solution for every point in time. Second, we analyze how well a k-optimal solution approximates an optimal solution. Typically we are not able to directly obtain good bounds on the approximation ratio, but given certain reasonable assumptions, good approximation bounds as a function of k can be provided.

3.2 Event Stability for EMSTs

Our input consists of a set of points $P = \{p_1, \ldots, p_n\}$ where each point p_i has a trajectory described by the function $x_i \colon [0, T] \to \mathbb{R}^d$. The goal is to maintain a combinatorial description of a short spanning tree on P that does not change often. We assume that the functions x_i are polynomials with bounded degree s.

To use the concept of k-optimal solutions, we first need to normalize the coordinates. We simply assume that $x_i(t) \in [0, 1]^d$ for $t \in [0, T]$. This assumption may seem overly restrictive for kinetic point sets, but note that we are only interested in relative positions, and thus the frame of reference may move with the points. Next, we define the metric $d_{\mathcal{I}}$ along the trajectory as follows.

$$d_{\mathcal{I}}(t, t') = \max_i \|x_i(t) - x_i(t')\| \tag{2}$$

Note that this metric, and the resulting definition of k-optimal solutions, is not specific to EMSTs and can be used in general for problems with kinetic point sets as input. In our case $\|a - b\|$ denotes the distance between a and b in the (Euclidean) ℓ_2 norm. Now let $OPT(t)$ be the EMST at time t. Then, by definition, $OPT(t)$ is k-optimal at time t' if $d_{\mathcal{I}}(t, t') \leq k$. Our approach is now very simple: we compute the EMST and keep that solution as long as it is k-optimal, after which we compute the new EMST, and so forth. Below we analyze this approach.

Number of events. To bound the number of events, we first need to bound the speed of any point with a polynomial trajectory and bounded coordinates. For this we can use a classic result known as the *Markov Brothers' inequality*.

Lemma 1 **([17]).** *Let $h(t)$ be a polynomial with degree at most s such that $h(t) \in [0, 1]$ for $t \in [0, T]$, then $|h'(t)| \leq s^2/T$ for all $t \in [0, T]$.*

Lemma 2. *For a kinetic point set P with degree-s polynomial trajectories $x_i(t) \in [0,1]^d$ ($t \in [0,T]$) we need only $O(\frac{s^2}{k})$ changes to maintain a k-optimal solution for constant d.*

Proof. By Lemma 1 the velocity of any point is at most s^2/T in one dimension, and thus at most $\sqrt{d}\, s^2/T = O(s^2/T)$ in d dimensions, assuming d is constant. Now assume that we have computed an optimal solution S for some time t. The solution S remains k-optimal until one of the points has moved at least k units. Since the velocity of the points is bounded, this takes at least $\Delta t = kT/s^2$ time, at which point we can recompute the optimal solution. Since the total time interval is of length T, this can happen at most $T/\Delta t = s^2/k$ times. □

Approximation factor. We cannot expect k-optimal solutions to be a good approximation of an optimal EMST's length in general: if all points are within distance k from each other, then all solutions are k-optimal. We therefore need to make the assumption that the points are spread out reasonably throughout the motion. To quantify this, we use a measure inspired by the *order-l spread*, as defined in [11]. Let $\textsc{mindist}_l(P)$ be the smallest distance in P between a point and its l-th nearest neighbor. We assume that $\textsc{mindist}_l(P) \geq 1/\Delta_l$ throughout the motion, for some value of Δ_l. We can use this assumption to give a lower bound on the length of the EMST. Pick an arbitrary point and remove all points from P that are within distance $1/\Delta_l$, and repeat this process until the smallest distance is at least $1/\Delta_l$. By our assumption, we remove at most $l-1$ points for each chosen point, so we are left with at least n/l points. The length of the EMST on P is at least the length of the EMST on the remaining n/l points, which has length $\Omega(\frac{n}{l\Delta_l})$.

Lemma 3. *A k-optimal solution of the EMST problem on a set of n points P is an $O(1 + kl\Delta_l)$-approximation of the EMST, under the assumption that $\textsc{mindist}_l(P) \geq 1/\Delta_l$.*

Proof. Let S be a k-optimal solution of P and let OPT be an optimal solution of P. By definition there is a point set P' for which the length of solution S is at most that of OPT. Since $d_{\mathcal{I}}(P, P') \leq k$, the length of each edge can grow or shrink by at most $2k$ when moving from P' to P. Therefore we can state that $f(P, S) \leq f(P, \mathrm{OPT}) + 4kn$. Now, using the lower bound on the length of an EMST, we obtain the following.

$$\begin{aligned} f(P, \mathrm{OPT}) + 4kn &\leq f(P, \mathrm{OPT}) + 4kO(f(P, \mathrm{OPT})l\Delta_l) \\ &= O(1 + kl\Delta_l) \cdot f(P, \mathrm{OPT}) \end{aligned}$$ □

Note that there is a clear tradeoff between the approximation ratio and how restrictive the assumption on the spread is. Regardless, we can obtain a decent approximation while only processing a small number of events. If we choose reasonable values $k = O(1/n)$, $l = O(1)$, and $\Delta_l = O(n)$, then our results show that, under the assumptions, a constant-factor approximation of the EMST can be maintained while processing only $O(n)$ events.

4 Topological Stability

The event stability analysis has two major drawbacks: (1) it is only applicable to problems for which the solutions are always feasible and described combinatorially, and (2) it does not distinguish between small and large structural changes. Topological stability analysis is applicable to a wide variety of problems and enforces continuous changes to the solution.

4.1 Topological Stability Analysis

Let Π be an optimization problem with input instances $\mathcal{I}$, solutions $\mathcal{S}$, and optimization function f. An algorithm $\mathcal{A}\colon \mathcal{I} \rightarrow \mathcal{S}$ is *topologically stable* if, for any (continuous) path $\pi\colon [0,1] \rightarrow \mathcal{I}$ in $\mathcal{I}$, $\mathcal{A}\pi$ is a (continuous) path in $\mathcal{S}$. To properly define a (continuous) path in $\mathcal{I}$ and $\mathcal{S}$ we need to specify a topology $\mathcal{T}_{\mathcal{I}}$ on $\mathcal{I}$ and a topology $\mathcal{T}_{\mathcal{S}}$ on $\mathcal{S}$. Alternatively we could specify metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, but this is typically more involved. We then want to analyze the approximation ratio of any topologically stable algorithm with respect to OPT. That is, we are interested in the ratio

$$\rho_{\mathrm{TS}}(\Pi, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) = \inf_{\mathcal{A}} \sup_{I \in \mathcal{I}} \frac{f(I, \mathcal{A}(I))}{f(I, \mathrm{OPT}(I))} \tag{3}$$

where the infimum is taken over all topologically stable algorithms. Naturally, if OPT is already topologically stable, then this type of analysis does not provide any insight and the ratio is simply 1. However, in many cases, OPT is not topologically stable. The above analysis can also be applied if the solution space (or the input space) is discrete. In such cases, continuity can often be defined using the graph topology of so-called flip graphs, for example, based on edge flips for triangulations or rotations in rooted binary trees. We can represent a graph as a topological space by representing vertices by points, and representing every edge of the graph by a copy of the unit interval $[0,1]$. These intervals are glued together at the vertices. In other words, we consider the corresponding simplicial 1-complex. Although the points in the interior of the edges of this topological space do not represent proper spanning trees, we can still use this topological space in Eq. 3 by extending f over the edges via linear interpolation. It is not hard to see that we need to consider only the vertices of the flip graph (which represent proper spanning trees) to compute the topological stability ratio.

4.2 Topological Stability of EMSTs

We use the same setting of the kinetic EMST problem as in Sect. 3.2, except that we do not restrict the trajectories of the points and we do not normalize the coordinates. We merely require that the trajectories are continuous. To define this properly, we need to define a topology on the input space, but for a kinetic point set with n points in d dimensions we can simply use the standard topology on $\mathbb{R}^{dn}$ as $\mathcal{T}_{\mathcal{I}}$. To apply topological stability analysis, we also need to specify a topology on the (discrete) solution space. As the points move, the minimum spanning tree may have to change at some point in time by removing one edge and inserting another edge. Since these two edges may be very far apart, we do not consider this operation to be stable or continuous. Instead we specify the topology of $\mathcal{S}$ using a flip graph, where the operations are either *edge slides* or *edge rotations* [1,12]. The optimization function f, measuring the quality of the EMST, is naturally defined for the vertices of the flip graph as the length of the spanning tree, and we use linear interpolation to define f on the edges of the flip graph. For edge slides and rotations we provide upper and lower bounds on $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}})$.

Edge slides. An edge slide is defined as the operation of moving one endpoint of an edge to one of its neighboring vertices along the edge to that neighbor. More formally, an edge (u, v) in the tree can be replaced by (u, w) if w is a neighbor of v and $w \neq u$. Since this operation is very local, we consider it to be stable. Note that after every edge slide the tree must still be connected.

Lemma 4. *If $\mathcal{T}_{\mathcal{S}}$ is defined by edge slides, then $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \leq \frac{3}{2}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now slide edge e to edge e' by sliding it along the vertices of C. Let x be the longest intermediate edge when sliding from e to e' (see Fig. 2(a)). To allow x to be as long as possible with respect to the length of the EMST, the EMST should be fully contained in C. By the triangle inequality we get that $2|x| \leq |C|$. Since the length of the EMST is $\mathrm{OPT} = |C| - |e|$, we get that $|x| \leq \mathrm{OPT}/2 + |e|/2$. Thus, the length of the intermediate tree is $|C| - 2|e| + |x| = \mathrm{OPT} - |e| + |x| \leq \frac{3}{2}\,\mathrm{OPT}$. □

Lemma 5. *If $\mathcal{T}_{\mathcal{S}}$ is defined by edge slides, then $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{\pi+1}{\pi}$.*

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e|$ is very small. Let the remaining points be arranged in a circle, as shown in Fig. 1(a), such that the farthest distance between any two points is $\mathrm{OPT}/\pi - \varepsilon$, where OPT is the length of the EMST. We can make this construction for any $\varepsilon > 0$ by using enough points and making e and e' arbitrarily short. Simply sliding e to e' will always grow e to be the diameter of the circle at some point. Alternatively, e can take a shortcut by sliding over another edge f as a chord (see Fig. 1(b)). This is only beneficial if $|e| + |f| < \mathrm{OPT}/\pi - \varepsilon$. However, if f helps e to avoid becoming a diameter of the circle, then e and f, as chords, must span an angle larger than π together. As a result, $|e| + |f| \geq \mathrm{OPT}/\pi - \varepsilon$ by triangle inequality.

A motion of the points that forces e to slide to e' in this particular configuration looks as follows. The points start at e and move at constant speed along the circle, half of the points clockwise and the other half counter clockwise. The speeds are assigned in such a way that at some point all points are evenly spread along the circle. Once all points are evenly spread, they start moving towards e', again along the circle. It is easy to see that, using the arguments above, any

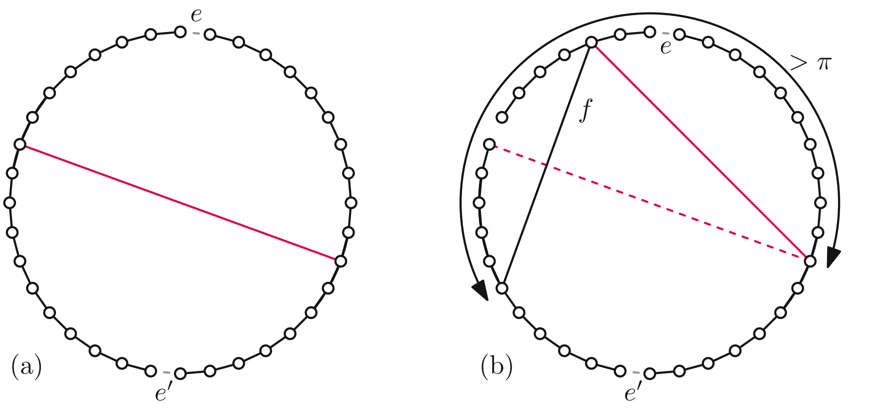


Fig. 1. This configuration is a $(\frac{\pi+1}{\pi} - \varepsilon)$-approximation of the EMST. (a) While e slides to e' it becomes the diameter of the circle. (b) Stretching an edge f to form a chord creates an even longer spanning tree.

additional edges inside the circle must have total length of at least the diameter of the circle at some point throughout the motion. On the other hand, OPT is largest when the points are evenly spread along the circle. Thus, for any $\varepsilon > 0$, $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{\pi+1}{\pi} - \varepsilon \approx 1.318 - \varepsilon$. □

Edge rotations. Edge rotations are a generalization of edge slides, that allow one endpoint of an edge to move to any other vertex. These operations are clearly not as stable as edge slides, but they are still more stable than the deletion and insertion of arbitrary edges.

Lemma 6. *If $\mathcal{T}_{\mathcal{S}}$ is defined by edge rotations, then $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \leq \frac{4}{3}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge $e = (u, v)$ and inserting an edge $e' = (u', v')$, where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now rotate edge e to edge e' along some of the vertices of C. Let x be the longest intermediate edge when rotating from e to e'. To allow x to be as long as possible with respect to the length of the EMST, the EMST should be fully contained in C. We argue that $|x| \leq \mathrm{OPT}/3 + |e|$, where OPT is the length of the EMST. Removing e and e' from C splits C into two parts, where we assume that u and u' (v and v') are in the left (right) part. First assume that one of the two parts has length at most $\mathrm{OPT}/3$. Then we can rotate e to (u, v'), and then to e', which implies that $|x| = |(u, v')| \leq \mathrm{OPT}/3 + |e|$ by the triangle inequality (see Fig. 2(b)). Now assume that both parts have length at least $\mathrm{OPT}/3$. Let $e_L = (u_L, v_L)$ be the edge in the left part that contains the midpoint of that part, and let $e_R = (u_R, v_R)$ be the edge in the right part that contains the midpoint of that part, where u_L and u_R are closest to e (see Fig. 2(c)). Furthermore, let Z be the length of $C \setminus \{e, e', e_L, e_R\}$. Now consider the potential edges (u, v_R), (v, v_L), (u', u_R), and (v', u_L). By the triangle inequality, the sum of the lengths of these

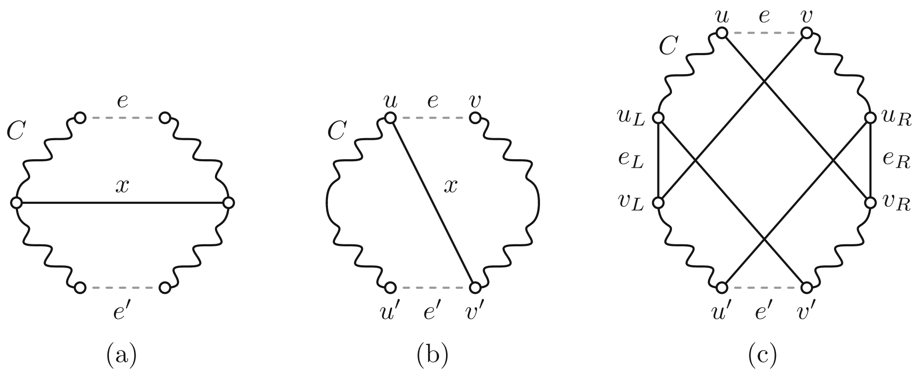
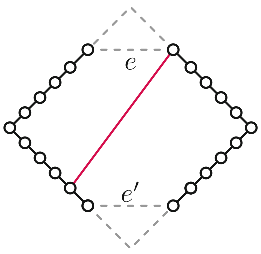


Fig. 2. (a) Illustration for Lemma 4. (b) and (c) Illustrating the two cases for Lemma 6.

edges is at most $4|e| + 2|e_L| + 2|e_R| + Z$. Thus, one of these potential edges has length at most $|e|+|e_L|/2+|e_R|/2+Z/4$. Without loss of generality let (u, v_R) be that edge (the construction is fully symmetric). We can now rotate e to (u, v_R), then to (u', v_R), and finally to e'. As each part of C has length at most $2\,\mathrm{OPT}/3$, we get that $|(u', v_R)| \leq \mathrm{OPT}/3 + |e|$ by construction. Furthermore we have that $\mathrm{OPT} = |e| + |e_L| + |e_R| + Z$. Thus, $|(u, v_R)| \leq |e| + |e_L|/2 + |e_R|/2 + Z/4 = \mathrm{OPT}/3 + 2|e|/3 + |e_L|/6 + |e_R|/6 - Z/12$. Since e needs to be removed to update the EMST, it must be the longest edge in C. Therefore $|(u, v_R)| \leq \mathrm{OPT}/3 + |e|$, which shows that $|x| \leq \mathrm{OPT}/3 + |e|$. Since the length of the intermediate tree is $\mathrm{OPT} - |e| + |x| \leq \frac{4}{3}\,\mathrm{OPT}$, we obtain that $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \leq \frac{4}{3}$. □

Lemma 7. *If $\mathcal{T}_{\mathcal{S}}$ is defined by edge rotations, then,* $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}}$.

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e'. Let the remaining points be arranged in a diamond shape as shown in the figure on the right, where the side length of the diamond is 2, and $|e| = |e'| = 1$. As a result, the distance between an endpoint of e and the left or right corner of the diamond is $2 - \frac{1}{2}\sqrt{2}$. Now we define a *top-connector* as an edge that intersects the vertical diagonal of the diamond, but is completely above the horizontal diagonal of the diamond.



A *bottom-connector* is defined analogously, but must be completely below the horizontal diagonal. Finally, a *cross-connector* is an edge that crosses or touches both diagonals of the diamond. Note that a cross-connector has length at least 2, and a top- or bottom-connector has length at least $|e| = 1$. In the considered update, we start with a top-connector and end with a bottom-connector. Since we cannot rotate from a top-connector to a bottom-connector in one step, we must reach a state that either has both a top-connector and a bottom-connector, or a single cross-connector. In both options the length of the spanning tree is $10 - 2\sqrt{2}$, while the minimum spanning tree has length $9 - 2\sqrt{2}$.

To force the update from e to e' in this configuration, we can use the following motion. The points start at the endpoints of e and move with constant speeds to a position where the points are evenly spread around the left and right corner of the diamond. Then the points move with constant speeds to the endpoints of e'. The argument above still implies that we need edges of total length at least 2 intersecting the vertical diagonal of the diamond at some point during the motion. On the other hand, $OPT \leq 9 - 2\sqrt{2}$ throughout the motion. Thus $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}} \approx 1.162$. □

5 Lipschitz Stability

The major drawback of topological stability analysis is that it still does not fully capture stable behavior; the algorithm must be continuous, but we can still make many changes to the solution in an arbitrarily small time step. In Lipschitz stability analysis we additionally limit how fast the solution can change.

5.1 Lipschitz Stability Analysis

Let Π be an optimization problem with input instances $\mathcal{I}$, solutions $\mathcal{S}$, and optimization function f. To quantify how fast a solution changes as the input changes, we need to specify metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ on $\mathcal{I}$ and $\mathcal{S}$, respectively. An algorithm $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$ is *K-Lipschitz stable* if for any $I, I' \in \mathcal{I}$ we have that $d_{\mathcal{S}}(\mathcal{A}(I), \mathcal{A}(I')) \leq K d_{\mathcal{I}}(I, I')$. We are then again interested in the approximation ratio of any K-Lipschitz stable algorithm with respect to OPT. That is, we are interested in the ratio

$$\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}}) = \inf_{\mathcal{A}} \sup_{I \in \mathcal{I}} \frac{f(I, \mathcal{A}(I))}{f(I, \mathrm{OPT}(I))} \tag{4}$$

where the infimum is taken over all K-Lipschitz stable algorithms. It is easy to see that $\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}})$ is lower bounded by $\rho_{\mathrm{TS}}(\Pi, \mathcal{T}_{\mathcal{I}}, \mathcal{T}_{\mathcal{S}})$ for the corresponding topologies $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{S}}$ of $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, respectively. As already mentioned in Sect. 2, analyses of this type are often quite hard. First, we often need to be very careful when choosing the metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$, as they should behave similarly with respect to scale. For example, let the input consist of a set of points in the plane and let cI for $I \in \mathcal{I}$ be the instance obtained by scaling all coordinates of the points in I by the factor c. Now assume that $d_{\mathcal{I}}$ depends linearly on scale, that is $d_{\mathcal{I}}(cI, cI') \sim c d_{\mathcal{I}}(I, I')$, and that $d_{\mathcal{S}}$ is independent of scale. Then, for some fixed K, we can reduce the effective speed of any K-Lipschitz stable algorithm arbitrarily by scaling down the instances sufficiently, rendering the analysis meaningless. We further need to be careful with discrete solution spaces. However, using the flip graphs as mentioned in Sect. 4 we can extend a discrete solution space to a continuous space by including the edges.

Typically it will be hard to fully describe $\rho_{\mathrm{LS}}(\Pi, K, d_{\mathcal{I}}, d_{\mathcal{S}})$ as a function of K. However, it may be possible to obtain interesting results for certain values of K. One value of interest is the value of K for which the approximation ratio equals or approaches the approximation ratio of the corresponding topological stability analysis. Another potential value of interest is the value of K below which any K-Lipschitz stable algorithm performs asymptotically as bad as a constant algorithm always computing the same solution regardless of instance.

5.2 Lipschitz Stability of EMSTs

We use the same setting of the kinetic EMST problem as in Sect. 4.2, except that, instead of topologies, we specify metrics for $\mathcal{I}$ and $\mathcal{S}$. For $d_{\mathcal{I}}$ we can simply use the metric in Eq. 2, which implies that points move with a bounded speed. For $d_{\mathcal{S}}$ we use a metric inspired by the edge slides of Sect. 4.2. To that end, we need to define how long a particular edge slide takes, or equivalently, how "far" an edge slide is. To make sure that $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ behave similarly with respect to scale, we let $d_{\mathcal{S}}$ measure the distance the sliding endpoint has traveled during an edge slide. However, this creates an interesting problem: the edge on which the endpoint is sliding may be moving and stretching/shrinking during the operation. This influences how long it takes to perform the edge slide. We need to be more specific: (1) as the points are moving, the relative position (between 0 and 1 from starting endpoint to finishing endpoint) of a sliding endpoint is maintained without cost in $d_{\mathcal{S}}$, and (2) $d_{\mathcal{S}}$ measures the difference in relative position multiplied by the length $L(t)$ of the edge on which the endpoint is sliding. More tangibly, an edge slide performed by a K-Lipschitz stable algorithm can be performed in t^* time such that $\int_0^{t^*} \frac{K}{L(t)} \mathrm{d}t = 1$, where $L(t)$ describes the length of the edge on which the endpoint slides as a function of time. Finally, the optimization function f simply computes a linear interpolation of the cost on the edges of the flip graph defined by edge slides.

We now give an upper bound on K below which any K-Lipschitz stable algorithm for kinetic EMST performs asymptotically as bad as any fixed tree. Given the complexity of the problem, our bound is fairly crude. We state it anyway to demonstrate the use of our framework, but we believe that a stronger bound exists. Note that any spanning tree is an $O(n)$-approximation of the EMST.

Lemma 8. *Let $d_{\mathcal{S}}$ be the metric for edge slides, then $\rho_{\mathrm{LS}}(EMST, \frac{c}{\log n}, d_{\mathcal{I}}, d_{\mathcal{S}}) = \Omega(n)$ for a small enough constant $c > 0$, where n is the number of points.*

6 Conclusion

We presented a framework for algorithm stability, which includes three types of stability analysis, namely event stability, topological stability, and Lipschitz stability. We also demonstrated the use of this framework by applying the different types of analysis to the kinetic EMST problem, deriving new interesting results.

We believe that, by providing different types of stability analysis with increasing degrees of complexity, we make stability analysis for algorithms more accessible, and make it easier to formulate interesting open problems on algorithm stability.

However, the framework that we presented does not (yet) give a complete picture: we do not consider the algorithmic aspect of stability. For example, if we already know how the input will change over time, can we efficiently compute a stable function of solutions over time that is optimal with regard to solution quality? Or, in a more restricted sense, can we efficiently compute the one solution that is best for all inputs over time? Even in the black-box model we can consider designing efficient algorithms that are K-Lipschitz stable and perform well with regard to solution quality. We leave such problems for future work.

References

1. Aichholzer, O., Aurenhammer, F., Hurtado, F.: Sequences of spanning trees and a fixed tree theorem. Comput. Geom. Theory Appl. **21**(1–2), 3–20 (2002)
2. Bacciotti, A., Rosier, L.: Liapunov Functions and Stability in Control Theory, 2nd edn. Springer, Heidelberg (2006). https://doi.org/10.1007/b139028
3. Basch, J., Guibas, L.J., Hershberger, J.: Data structures for mobile data. J. Algorithms **31**(1), 1–28 (1999)
4. 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)
5. Bousquet, O., Elisseeff, A.: Stability and generalization. J. Mach. Learn. Res. **2**, 499–526 (2002)
6. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D.P., Kobourov, S.G., Lubiw, A., Mitchell, J.S.: On simultaneous planar graph embeddings. Comput. Geom. Theory Appl. **36**(2), 117–130 (2007)
7. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. In: Proceedings of 21st Symposium on Computational Geometry, pp. 263–271 (2005)
8. de Berg, M., Roeloffzen, M., Speckmann, B.: Kinetic 2-centers in the black-box model. In: Proceedings of 29th Symposium on Computational Geometry, pp. 145–154 (2013)
9. Durocher, S., Kirkpatrick, D.: The Steiner centre of a set of points: stability, eccentricity, and applications to mobile facility location. Int. J. Comput. Geom. Appl. **16**(04), 345–371 (2006)
10. Durocher, S., Kirkpatrick, D.: Bounded-velocity approximation of mobile Euclidean 2-centres. Int. J. Comput. Geom. Appl. **18**(03), 161–183 (2008)
11. Erickson, J.: Dense point sets have sparse Delaunay triangulations or "... but not too nasty". Discret. Comput. Geom. **33**(1), 83–115 (2005)
12. Goddard, W., Swart, H.C.: Distances between graphs under edge operations. Discret. Math. **161**(1–3), 121–132 (1996)
13. Guibas, L.J.: Kinetic data structures. In: Mehta, D.P., Sahni, S. (eds.) Handbook of Data Structures and Applications, pp. 23.1–23.18. Chapman and Hall/CRC, Boca Raton (2004)
14. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia (2002)

15. Katoh, N., Tokuyama, T., Iwano, K.: On minimum and maximum spanning trees of linearly moving points. In: Proceedings of 33rd Symposium on Foundations of Computer Science, pp. 396–405 (1992)
16. Kitchin, R.M.: Cognitive maps: what are they and why study them? J. Environ. Psychol. **14**(1), 1–19 (1994)
17. Markoff, W.: Über Polynome, die in einem gegebenen Intervalle möglichst wenig von Null abweichen. Math. Ann. **77**(2), 213–258 (1916)
18. Monma, C., Suri, S.: Transitions in geometric minimum spanning trees. Discret. Comput. Geom. **8**(3), 265–293 (1992)
19. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J. ACM **51**(3), 385–463 (2004)

Rapid Mixing of k-Class Biased Permutations

Sarah Miracle[1(✉)] and Amanda Pascoe Streib[2]

[1] University of St. Thomas, St. Paul, MN 55105, USA
sarah.miracle@stthomas.edu
[2] Center for Computing Sciences, Bowie, MD 20715, USA
ampasco@super.org

Abstract. In this paper, we study a biased version of the nearest-neighbor transposition Markov chain on the set of permutations where neighboring elements i and j are placed in order (i, j) with probability $p_{i,j}$. Our goal is to identify the class of parameter sets $\mathbf{P} = \{p_{i,j}\}$ for which this Markov chain is rapidly mixing. Specifically, we consider the open conjecture of Jim Fill that all monotone, positively biased distributions are rapidly mixing.

We resolve Fill's conjecture in the affirmative for distributions arising from k-class particle processes, where the elements are divided into k classes and the probability of exchanging neighboring elements depends on the particular classes the elements are in. We further require that k is a constant, and all probabilities between elements in different classes are bounded away from 1/2. These particle processes arise in the context of self-organizing lists and our result also applies beyond permutations to the setting where all particles in a class are indistinguishable. Our work generalizes recent work by Haddadan and Winkler (STACS '17) studying 3-class particle processes. Additionally we show that a broader class of distributions based on trees is also rapidly mixing, which generalizes a class analyzed by Bhakta et al. (SODA '13).

Our proof involves analyzing a generalized biased exclusion process, which is a nearest-neighbor transposition chain applied to a 2-particle system. Biased exclusion processes are of independent interest, with applications in self-assembly. We generalize the results of Greenberg et al. (SODA '09) and Benjamini et al. (Trans. AMS '05) on biased exclusion processes to allow the probability of swapping neighboring elements to depend on the entire system, as long as the minimum bias is bounded away from 1.

1 Introduction

The fundamental problem of generating a random permutation has a long history in computer science, beginning as early as 1969 [1]. One way to generate a random permutation is to use the nearest-neighbor Markov chain $\mathcal{M}_{\text{nn}}$ which repeatedly swaps the elements in a random pair of adjacent positions. The chain $\mathcal{M}_{\text{nn}}$ was among the first considered in the study of the computational efficiency

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 820–834, 2018.
https://doi.org/10.1007/978-3-319-77404-6_59

of Markov chains for sampling [2–4] and has subsequently been studied extensively. After a series of papers, the mixing time of $\mathcal{M}_{\mathrm{nn}}$ ($\Theta(n^3 \log n)$ [5]) is now well-understood when sampling from the uniform distribution.

The nearest-neighbor chain $\mathcal{M}_{\mathrm{nn}}$ can also be used to sample from more general probability distributions on the permutation group S_n by allowing non-uniform swap probabilities. Suppose we have a set of parameters $\mathcal{P} = \{p_{i,j}\}$ and that $\mathcal{M}_{\mathrm{nn}}$ puts neighboring elements i and j in order (i, j) with probability $p_{i,j}$, where $p_{j,i} = 1 - p_{i,j}$. Despite the simplicity of this natural extension, much less is known about the mixing time of $\mathcal{M}_{\mathrm{nn}}$ in the non-uniform case. In this paper we look at the question for which parameter sets $\mathcal{P}$ is $\mathcal{M}_{\mathrm{nn}}$ rapidly (polynomially) mixing? We say $\mathcal{P}$ is *positively biased* if $p_{i,j} \geq 1/2$ for all $i < j$. Without this condition, it is fairly straightforward to construct parameter sets for which $\mathcal{M}_{\mathrm{nn}}$ has exponential mixing time (see e.g., [6]). Interestingly, Bhakta et al. [6] showed that $\mathcal{M}_{\mathrm{nn}}$ can require exponential time to mix even for distributions with positive bias. In a widely circulated manuscript, Fill [7,8] introduced the following monotonicity conditions: $p_{i,j} \leq p_{i,j+1}$ and $p_{i,j} \geq p_{i+1,j}$ for all $1 \leq i < j \leq n$. Fill conjectured that $\mathcal{M}_{\mathrm{nn}}$ is rapidly mixing for all monotone, positively biased distributions and further conjectured that the smallest spectral gap for $\mathcal{M}_{\mathrm{nn}}$ is given by the uniform $p_{i,j} = 1/2$ distribution. He confirmed these conjectures for $n \leq 3$ and gave experimental evidence for $n \leq 5$.

Except for a few special classes of monotone, positively biased distributions very little is known about the mixing time of $\mathcal{M}_{\mathrm{nn}}$ in the non-uniform setting and the conjecture has remained unproven for over a decade. Benjamini et al. [9] studied the case that $p_{i,j} = p > 1/2$ for all $i < j$. They showed that the mixing time of $\mathcal{M}_{\mathrm{nn}}$ is $\Theta(n^2)$ for these distributions. Bhakta et al. [6] showed that if $p_{i,j}$ depends on only the smaller of i and j, then $\mathcal{M}_{\mathrm{nn}}$ mixes in polynomial time. They extend this to distributions arising from binary trees with leaves labeled $\{1, 2, \ldots, n\}$ and internal nodes labeled by probabilities, where the label of the lowest common ancestor of i and j in the tree determines $p_{i,j}$.

Recently there has been interest in a special class of monotone, positively biased distributions highlighted by Fill [8]. These distributions are motivated by self-organizing lists and closely related to exclusion processes arising in statistical physics. In this setting the set $[n]$ is partitioned into k classes $C_1, C_2, \ldots, C_k$ and the probability of swapping two elements in $[n]$ is determined by the classes containing those elements. Define a k-class as a set of probabilities $\mathcal{P}$ where elements from the same set are exchanged with probability $1/2$ while each element from C_i and each element from C_j (with $i < j$) are put in increasing order with the same probability $p_{i,j} > 1/2$. The setting of k-classes is motivated by self-organizing lists (see [10] for a survey). Consider a set of records in a linear array, where element i is requested with some unknown frequency w_i. A self-organizing list is a way to reduce the linear look-up time by adjusting the permutation each time an element is requested. The Move-Ahead-One (also called Transpose) algorithm updates the permutation by moving the element forward one position; i.e. it is precisely the nearest neighbor transposition chain. The probability of swapping i and j is $p_{i,j} = w_i/(w_i + w_j)$. We call this set of distributions w-distributions [8], or more explicitly k-value w-distributions (where there are k distinct frequencies w_i). Note these are an example of k-classes.

While w-distributions are quite natural and appear to be rapidly mixing [8], this particular simple instance of the biased permutation problem has so far eluded a thorough analysis. Haddadan and Winkler [11] recently studied 3-value w-distributions. They showed if $w_2/w_3, w_1/w_2 \geq 2$, then $\mathcal{M}_{\text{nn}}$ has mixing time $O(n^{18})$. They also analyzed a related nearest neighbor chain $\mathcal{M}_{\text{pp}}$ over 3-particle systems, where the elements within a class are indistinguishable, and so they are never swapped. They showed the mixing time of this chain is at most $O(n^{10})$.

Our Main Result. Here we consider bounded k-classes, where $p_{i,j}/p_{j,i} \geq \gamma$ for all $i < j$ for some constant $\gamma > 1$. We show that if $\mathcal{P}$ is a weakly monotone bounded k-class then the mixing time of $\mathcal{M}_{\text{nn}}$ is $O(n^{2k+6} \log k)$. This gives a polynomial bound for any constant k, and applies directly to all bounded k-value w-distributions (i.e. $w_i/w_{i+1} > \gamma$ for all i). This improves the mixing time bound given in [11] for $k = 3$. We also analyze $\mathcal{M}_{\text{pp}}$ over k-particle systems, and find the mixing time is $O(n^{2k+4})$, matching the bounds from [11] for $k = 3$. In both cases, we extend their results to allow $\gamma < 2$. In addition, we extend the work of Bhakta et al. [6] on distributions based on binary trees to include trees with maximum degree at most k.

Biased Exclusion Processes. Simple 2-class particle systems, known as biased exclusion processes, have been a key tool in the study of biased permutations. Suppose there are two types of particles (say 1 and 0) on a line, with n_i (indistinguishable) particles of type i. Define a (finite) biased exclusion process over the linear arrangements of these particles as follows: at each step, a pair of neighboring particles of different types may swap into increasing order with probability p or out of order with probability $1 - p$. Much of the previous work on the biased permutation problem has proceeded by mapping $\mathcal{M}_{\text{nn}}$ over permutations to several biased exclusion processes or the related infinite asymmetric simple exclusion processes (ASEPs). In [9], $\mathcal{M}_{\text{nn}}$ is analyzed as a cross-product of several ASEPs, and then rapid mixing for $\mathcal{M}_{\text{nn}}$ is inferred from the mixing times of the ASEPs. Bhakta et al. [6] discovered a different decomposition of permutations into a cross-product of biased exclusion processes, which allowed them to prove rapid mixing for more general $\mathcal{P}$ distributions.

Exclusion processes are of independent interest, arising in a variety of contexts. The infinite version known as the asymmetric simple exclusion process is a fundamental stochastic model in statistical mechanics [9,12]. In combinatorics, the unbiased exclusion process is known as the mountain/valley Markov chain over monotonic lattice paths (i.e. staircase walks) (see Fig. 1 and, e.g. [13]). Notice each linear arrangement of 1's and 0's can be mapped bijectively to a lattice path in $\mathbb{Z}^2$ by sending 1's to steps down and 0's to steps to the right. A biased version of this chain has applications in self-assembly, where it represents reversible growth processes [14,15].

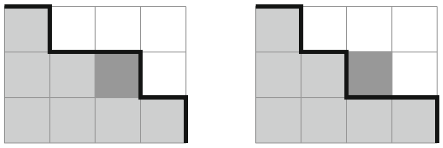


Fig. 1. Staircase walks that differ by a mountain/valley move (the darkened square).

Benjamini et al. [9] bounded the mixing time of the asymmetric exclusion process, where particles of type 0 and type 1 all interact with the same (constant) probability p. Subsequently, Greenberg et al. [14] discovered a simpler proof. This continues to be an active area of interest and in recent work Labbé and Lacoin [16] determined the exact mixing rate and Levin and Peres [17] analyzed the case that p tends to 0 as $n \to \infty$. Greenberg et al. [15,18] considered a heterogeneous biased exclusion process, where the probability of swapping a 1 with a 0 at positions i and $i+1$ depends on the numbers of 1's and 0's to the left of position i.

In this paper, we introduce a new *generalized (biased) exclusion process*, where the probability of swapping a 1 with a 0 may depend on the entire sequence of 0's and 1's and prove it is rapidly mixing whenever the minimum bias is at least a constant. Analyzing these processes is a key step towards proving our main result on permutations, and we believe it could be of interest beyond the application to biased permutations.

Techniques. In order to analyze $\mathcal{M}_{\text{nn}}$ we introduce a new Markov chain $\mathcal{M}_{\text{T}}$ which includes all transitions of $\mathcal{M}_{\text{nn}}$ plus a carefully selected set of more general transpositions (swaps between non-nearest neighbor pairs). We will then infer rapid mixing of $\mathcal{M}_{\text{nn}}$ from the rapid mixing of $\mathcal{M}_{\text{T}}$. The new chain may be viewed as a combination of multiple unbiased permutation processes (one for each of the k classes) and a single biased k-particle process where elements in the same class are indistinguishable. The bulk of our work is in proving that the particle process is rapidly mixing. Here our argument relies on a novel decomposition argument where we fix the location of all of the particles in a single class and repeat this process inductively. By doing this, we can again simplify $\mathcal{M}_{\text{nn}}$ over permutations to several 2-class particle processes as in previous work [6,9], with two key differences. First, we reduce to our new generalized biased exclusion processes mentioned above, where the probability of swapping two particles depends on the entire state of the system. Second, we need to use a decomposition theorem [19,20] since in general $\mathcal{M}_{\text{nn}}$ does not appear to be a simple cross-product of a set of 2-class particle processes for monotone positively biased distributions. In fact, we use decomposition inductively $O(k)$ times.

2 The Markov Chains $\mathcal{M}_{\text{nn}}$ and $\mathcal{M}_{\text{T}}$

We begin by formalizing the Markov chain $\mathcal{M}_{\text{nn}}$. Then we will formally define a k-class and introduce an auxiliary chain $\mathcal{M}_{\text{T}}$ that allows a larger set of transpositions. Let $\Omega = S_n$ be the set of all permutations $\sigma = (\sigma(1), \ldots, \sigma(n))$ of n integers. Suppose $\mathcal{P}$ is a set of probabilities, consisting of $p_{i,j} \in [0,1]$ for each $1 \leq i \neq j \leq n$, where $p_{j,i} = 1 - p_{i,j}$.

The Nearest Neighbor Markov chain $\mathcal{M}_{\mathbf{nn}}$

`Starting at any permutation` σ_0`, iterate the following:`

- At time t, choose a position $1 < i \leq n$ uniformly at random.
- With probability $p_{\sigma_t(i),\sigma_t(i-1)}/2$, exchange the elements $\sigma_t(i)$ and $\sigma_t(i-1)$ to obtain σ_{t+1}.
- Otherwise, do nothing so that $\sigma_{t+1} = \sigma_t$.

The chain $\mathcal{M}_{\text{nn}}$ connects the state space Ω and has the stationary distribution (see e.g., [6]) $\pi(\sigma) = \left(\prod_{i<j} p_{\sigma(i),\sigma(j)}\right) Z^{-1}$, where Z is a normalizing constant.

For our main result we prove that if a set of probabilities $\mathcal{P}$ are weakly monotonic and form a bounded k-class then the Markov chain $\mathcal{M}_{\text{nn}}$ is rapidly mixing. We will require the weakly monotonic condition defined in [6] rather than the stronger monotonic condition defined in Fill's conjecture [7,8].

Definition 1 ([6]). *The set $\mathcal{P}$ is weakly monotonic if properties 1 and either 2 or 3 are satisfied.*

1. $p_{i,j} \geq 1/2$ *for all* $2 \leq i < j \leq n$*, and*
2. $p_{i,j+1} \geq p_{i,j}$ *for all* $1 \leq i < j \leq n-1$ *or*
3. $p_{i-1,j} \geq p_{i,j}$ *for all* $2 \leq i < j \leq n$*.*

For simplicity, we will assume that property (2) holds. If instead property (3) holds, the proofs are very similar and we point out distinctions as necessary.

Suppose $[n]$ is partitioned into k particle classes $\{C_i\}$. A set of probabilities forms a k-class if particles in the same class interact with probability 1/2 and the probability of swapping a particle in class C_i with a (neighboring) particle in class C_j is the same for all particles within those classes[1]: for all $1 \leq i < j \leq k$, if $x_1, x_2 \in C_i$ and $y \in C_j$ we have $p_{x_1,y} = p_{x_2,y}$. We associate each permutation σ with a *k-particle system*, where particles within the same class are indistinguishable and therefore given the same label. The k-particle system associated to σ is a sequence of n labels from $[k]$, where the i^{th} label is j if $\sigma(i) \in C_j$. Note that several permutations are associated with the same particle system. For example, if $C_1 = 1, C_2 = 2,3, C_3 = 4,5$, then both permutations 32145 and 23145 are associated with the k-particle system 22133. For any element x, let $C(x)$ denote the particle class that contains x (i.e. $C(x) = i$ if and only if $x \in C_i$). Let $C_i > C_j$ if $i > j$ and similarly for $C_i = C_j$. We say a k-class is *bounded* if there exists a constant $\gamma > 1$ such that for all $1 \leq i < j \leq n$, if $C(i) \neq C(j)$ then $p_{i,j}/p_{j,i} \geq \gamma$.

Next we define a non-nearest neighbor Markov chain $\mathcal{M}_{\text{T}}$: $\mathcal{M}_{\text{T}}$ exchanges elements $\sigma(i)$ and $\sigma(j)$ at locations i and j with $i < j$ if for all $i < m < j$,

[1] We assume, in order to ensure the distribution is positively biased, that for all $i \leq k$, $C_i = \{a_{i-1}+1, a_{i-1}+2, \ldots, a_i\}$, for some sequence $0 = a_0 < a_1 < a_2 < \ldots < a_{k-1}$.

$C(\sigma(m)) < \min(C(\sigma(i)), C(\sigma(j)))$. That is, $\mathcal{M}_{\mathrm{T}}$ swaps elements in different particle classes across elements in particle classes that are smaller than both. Particles in the same class can also be exchanged across any particles in other classes. For example, suppose you start from a permutation with associated k-particle system 3152673. The chain $\mathcal{M}_{\mathrm{T}}$ would allow an exchange of the two 3 particles because they are in the same class and there are no other particles in class 3 between them. An exchange of the first 3 and 5 (resulting in 5132673) would be allowed but an exchange of 5 and 7 would not because there is an element in class 6 between them. Let $\lambda_{i,j} = p_{i,j}/p_{j,i}$.

The Transposition Markov chain $\mathcal{M}_{\mathrm{T}}$

`Starting at any permutation` σ_0`, iterate the following:`

- At time t, choose $1 \leq i \leq n$ and $d \in \{L, R, N\}$ uniformly at random.
- If $d = L$, find the largest j with $1 \leq j < i$ and $C(\sigma_t(j)) \geq C(\sigma_t(i))$ (if one exists). If $C(\sigma_t(j)) > C(\sigma_t(i))$, then with probability 1/2, exchange $\sigma_t(i)$ and $\sigma_t(j)$ to obtain σ_{t+1}.
- If $d = R$, find the smallest j with $n \geq j > i$ and $C(\sigma_t(j)) \geq C(\sigma_t(i))$ (if one exists). If $C(\sigma_t(j)) > C(\sigma_t(i))$, then with probability
$$\frac{1}{2}\,\lambda_{\sigma_t(j),\sigma_t(i)} \prod_{i<k<j} \left(\lambda_{\sigma_t(j),\sigma_t(k)}\lambda_{\sigma_t(k),\sigma_t(i)}\right),$$
exchange $\sigma_t(i)$ and $\sigma_t(j)$ to obtain σ_{t+1}.
- If $d = N$, find the largest j with $1 \leq j < i$ and $C(\sigma_t(j)) = C(\sigma_t(i))$. If such an element exists, then with probability 1/2, exchange the elements $\sigma_t(i)$ and $\sigma_t(j)$ to obtain σ_{t+1}.
- Otherwise, do nothing so that $\sigma_{t+1} = \sigma_t$.

It is a simple exercise to show that $\mathcal{M}_{\mathrm{T}}$ samples from the same distribution as $\mathcal{M}_{\mathrm{nn}}$ and the proof is deferred to the final version of this paper.

The time a Markov chain takes to converge to its stationary distribution, or *mixing time*, is measured in terms of the distance between the distribution at time t and the stationary distribution. The *total variation distance* at time t is $\|P^t, \pi\|_{tv} = \max_{x\in\Omega} \frac{1}{2}\sum_{y\in\Omega} |P^t(x,y) - \pi(y)|$, where $P^t(x,y)$ is the t-step transition probability. For all $\epsilon > 0$, the *mixing time* $\tau(\epsilon)$ of $\mathcal{M}$ is defined as $\tau(\epsilon) = \min\{t : \|P^{t'}, \pi\|_{tv} \leq \epsilon, \forall t' \geq t\}$. We say that a Markov chain is *rapidly mixing* if the mixing time is bounded above by a polynomial in n and $\log(\epsilon^{-1})$, where n is the size of each configuration in Ω.

In our proofs we will bound the eigenvalue gaps of the transition matrices of $\mathcal{M}_{\mathrm{T}}$ and of $\mathcal{M}_{\mathrm{nn}}$. Let $Gap(P) = 1 - |\lambda_1|$ denote the spectral gap, where $\lambda_0, \lambda_1, \ldots, \lambda_{|\Omega|-1}$ are the eigenvalues of the transition matrix P and $1 = \lambda_0 > |\lambda_1| \geq |\lambda_i|$ for all $i \geq 2$. The following result relates the spectral gap with the mixing time of the chain (see, e.g., [21,22]):

Theorem 1 [22]**.** *Let* $\pi_* = \min_{x\in\Omega}\pi(x)$*. For all* $\epsilon > 0$ *we have*

$$(a) \qquad \tau(\epsilon) \leq \frac{1}{1-|\lambda_1|}\log\left(\frac{1}{\pi_*\epsilon}\right).$$

$$(b) \qquad \tau(\epsilon) \geq \frac{|\lambda_1|}{2(1-|\lambda_1|)}\log\left(\frac{1}{2\epsilon}\right).$$

In the remainder of the paper we prove $\mathcal{M}_{\mathrm{T}}$ and $\mathcal{M}_{\mathrm{nn}}$ are rapidly mixing if the input probabilities $\mathcal{P}$ are weakly monotonic and form a bounded k-class.

3 Bounded Generalized Exclusion Processes Mix Rapidly

We begin by analyzing bounded generalized biased exclusion processes. Assume n_1 particles of type 1 and n_0 particles of type 0 occupy n_0+n_1 linear positions: $1,\ldots,n_0+n_1$. Let Ω_e be the set of all distinct orderings of n_1 1's and n_0 0's. In this setting, the probabilities $p_{\sigma_t,i}$ depend on both the current ordering σ_t and the elements being exchanged. Consider the following chain on Ω_e.

The Generalized Exclusion Markov chain $\mathcal{M}_{\mathbf{ex}}$

`Starting at any configuration` σ_0`, iterate the following:`

- At time t, choose a position $1 \leq i < n_0+n_1$ uniformly at random.
- If $\sigma_t(i) \neq \sigma_t(i+1)$, with probability $p_{\sigma_t,i}$ exchange elements $\sigma_t(i)$ and $\sigma_t(i+1)$ to obtain σ_{t+1}.
- Otherwise, do nothing so that $\sigma_{t+1} = \sigma_t$.

We say that $\mathcal{M}_{\mathrm{ex}}$ is *bounded* if there exists a constant $\gamma > 1$ such that for all $\sigma \in \Omega_e$, if $\sigma(i) = 1$ and $\sigma(i+1) = 0$ and τ is obtained from σ by swapping elements $\sigma(i)$ and $\sigma(i+1)$, then $p_{\sigma,i}/p_{\tau,i} \geq \gamma$.

Recall the bijection between Ω_e and staircase walks: map 1's to steps down and 0's to steps to the right. For example, the two walks in Fig. 1 map to 0100101 and 0101001 respectively. Exchanging a 1 and a 0 corresponds to adding or removing a particular square beneath the staircase walk. Greenberg and others [14,15,18] studied a biased version of the "mountain/valley" chain for sampling staircase walks that start at $(0,h)$ and end at $(w,0)$. In [15,18], they assumed the surface has "fluctuating bias," meaning that each square s (on the $h\times w$ lattice) is assigned a bias λ_s which is essentially the ratio of the probabilities of adding or removing that particular square. They showed that as long as the minimum bias is a constant larger than 1 then the chain is rapidly mixing. In our setting, the probability of adding or removing a particular square can vary depending on the rest of the configuration. For example, the probability of moving from 1010 to 1001 is not necessarily the same as the probability of moving from 0110 to 0101. We prove the following theorem.

Theorem 2. *Let* $\mathcal{M}_{ex}$ *be a bounded generalized exclusion process on* n_1 *1's and* n_0 *0's. Suppose without loss of generality that* $n_1 \leq n_0$*. Then the mixing time* τ_{ex} *of* $\mathcal{M}_{ex}$ *satisfies* $\tau_{ex}(\epsilon) = O\left((n_0+n_1)\left(n_1 + \ln n_0 + \ln\epsilon^{-1}\right)\right)$.

Our proof is similar to that of [18] and we defer it to the final version. The idea is that the hitting time (time to reach the most probable configuration) yields a bound on the mixing time, and if the minimum bias is a constant, then the hitting time is on the order of the area of the region.

4 $\mathcal{M}_{\mathrm{T}}$ Mixes Rapidly for k-Class Biased Permutations

Next we prove that if the probabilities $\mathcal{P}$ form a k-class then the Markov chain $\mathcal{M}_{\mathrm{T}}$ mixes rapidly. This will be useful when we analyze the nearest neighbor chain $\mathcal{M}_{\mathrm{nn}}$ in Sect. 5. We first notice that the chain $\mathcal{M}_{\mathrm{T}}$ is a product of $k+1$ independent Markov chains ${\mathcal{M}_{\mathrm{T}}}^{(1)}, {\mathcal{M}_{\mathrm{T}}}^{(2)}, \ldots, {\mathcal{M}_{\mathrm{T}}}^{(k)}$ and $\mathcal{M}_{\mathrm{T}}^{*}$. The first k chains involve moves between particles in the same class and each such ${\mathcal{M}_{\mathrm{T}}}^{(i)}$ is an unbiased nearest-neighbor Markov chain over permutations of $|C_i|$ particles (i.e. moves of $\mathcal{M}_{\mathrm{T}}$ with direction N). The final chain $\mathcal{M}_{\mathrm{T}}^{*}$ allows only moves between different particle classes. We give the formal definitions of these chains below.

The Unbiased Markov chain $\mathcal{M}_{\mathbf{T}}^{(i)}$ (for $1 \leq i \leq k$)

`Starting at any permutation` σ_0`, iterate the following:`

- At time t, choose a position f with $C(\sigma_t(f)) = i$ uniformly at random.
- Find the largest g with $1 \leq g < f$ and $C(\sigma_t(g)) = C(\sigma_t(f)) = i$. If such an element exists, then with probability $1/2$ exchange the elements $\sigma_t(f)$ with $\sigma_t(g)$ to obtain σ_{t+1}.
- Otherwise, do nothing so that $\sigma_{t+1} = \sigma_t$.

The final chain $\mathcal{M}_{\mathrm{T}}^{*}$ allows moves between different particle classes. Note that $\mathcal{M}_{\mathrm{T}}^{*}$ includes all moves of $\mathcal{M}_{\mathrm{T}}$ except those with direction N.

The Markov chain $\mathcal{M}_{\mathbf{T}}^{*}$

`Starting at any permutation` σ_0`, iterate the following:`

- At time t, choose a position $1 \leq i \leq n$ and direction $d \in \{L, R\}$ uniformly at random.
- If $d = L$, find the largest j with $1 \leq j < i$ and $C(\sigma_t(j)) \geq C(\sigma_t(i))$ (if one exists). If $C(\sigma_t(j)) > C(\sigma_t(i))$, then with probability $1/2$ exchange the elements $\sigma_t(i)$ and $\sigma_t(j)$ to obtain σ_{t+1}.
- If $d = R$, find the smallest j with $n \geq j > i$ and $C(\sigma_t(j)) \geq C(\sigma_t(i))$ (if one exists). If $C(\sigma_t(j)) > C(\sigma_t(i))$, then with probability

$$\frac{1}{2}\,\lambda_{\sigma_t(j),\sigma_t(i)} \prod_{i<k<j} \left(\lambda_{\sigma_t(j),\sigma_t(k)}\lambda_{\sigma_t(k),\sigma_t(i)}\right),$$

  exchange $\sigma_t(i)$ and $\sigma_t(j)$ to obtain σ_{t+1}.
- Otherwise, do nothing so that $\sigma_{t+1} = \sigma_t$.

Theorem 3. *If the probabilities $\mathcal{P}$ are weakly monotonic and form a bounded k-class for $k \geq 2$, then the mixing time τ_T of $\mathcal{M}_T$ satisfies $\tau_T(\epsilon) = O\left(n^{2k}\ln(k\epsilon^{-1})\right)$.*

To prove Theorem 3, we use a result of [6] to relate the mixing times of the smaller chains $\{\mathcal{M}_{\mathrm{T}}{}^{(i)}\}_{i=1}^{k+1}$ to $\mathcal{M}_{\mathrm{T}}$. Previous results [5] allow us to bound the mixing times of $\mathcal{M}_{\mathrm{T}}{}^{(1)}, \mathcal{M}_{\mathrm{T}}{}^{(2)}, \ldots, \mathcal{M}_{\mathrm{T}}{}^{(k)}$. Thus, the bulk of our work is to bound the mixing time of $\mathcal{M}_{\mathrm{T}}^*$, which we do next.

4.1 *k*-Particle Processes Mix Rapidly

Recall $\mathcal{M}_{\mathrm{T}}^*$ allows only those moves of $\mathcal{M}_{\mathrm{T}}$ that involve elements in different particle classes (i.e. the moves with direction L and R). We call it a *k-particle process* over its state space of k-particle systems, since in this context elements in the same class are indistinguishable. If there are only two particle classes then this chain is a bounded generalized exclusion process. We prove the following.

Lemma 4. *Assume $|C_i| = c_i$ for all i. The spectral gap Gap(P) of the chain $\mathcal{M}_T^*$ satisfies Gap$(P) = \Omega\left(\left(n^{k-1}\prod_{i=1}^{k-1}(c_i + \ln n)\right)^{-1}\right)$. The mixing time τ_{T^*} of $\mathcal{M}_T^*$ satisfies $\tau_{T^*}(\epsilon) \leq O(n^{2k}\ln\epsilon^{-1})$.*

Our proof will proceed inductively, and at each step of the induction we will apply the decomposition theorem [19,23]. We will use the following version due to Martin and Randall [19]. Let $\Omega = \cup_{i=1}^m \Omega_i$ be a partition of the state space into m disjoint pieces. For each $i = 1, \ldots, m$, define $P_i = P(\Omega_i)$ as the restriction of P to Ω_i which rejects moves that leave Ω_i. In particular, the restriction to Ω_i is a Markov chain $\mathcal{M}_i$ with state space Ω_i, where the transition matrix P_i is defined as follows: If $x \neq y$ and $x, y \in \Omega_i$ then $P_i(x,y) = P(x,y)$; if $x \in \Omega_i$ then $P_i(x,x) = 1 - \sum_{y\in\Omega_i, y\neq x} P_i(x,y)$. Let π_i be the normalized restriction of π to Ω_i, i.e., $\pi_i(A) = (\pi(A\cap\Omega_i))/(\pi(\Omega_i))$. Define $\widehat{P}$ to be the following aggregated transition matrix on the state space $\{1, \ldots, m\}$: $\widehat{P}(i,j) = \frac{1}{\pi(\Omega_i)}\sum_{\substack{x\in\Omega_i,\\ y\in\Omega_j}} \pi(x)P(x,y)$.

Theorem 5 **([19]).** *Gap$(P) \geq \frac{1}{2}$Gap$(\widehat{P})\min_{i=1,\ldots,m}$ Gap(P_i).*

We may now prove Lemma 4. As a running example, let $C_1 = \{1,2\}, C_2 = \{3\}, C_3 = \{4,5\}, C_4 = \{6\}$, and $C_5 = \{7\}$. Since elements within a class are indistinguishable to $\mathcal{M}_{\mathrm{T}}^*$, we list each element using the subscript of its class; e.g., one 4-class particle system with these parameters is 4123315.

Proof. For $i \geq 0$, let σ_i represent an arbitrary fixed location of the particles in classes $C_1, C_2, \ldots, C_i$ (when $i = 0$, σ_i represents no restriction). For example, $\sigma_{k-2} =$ _12__1_, where the _ represents an empty location which will be filled by an element of C_{k-1} or C_k. We will consider a smaller chain $\mathcal{M}_{\sigma_i}$ whose state space is the set of all configurations where the elements in classes $C_1, \ldots, C_i$ are in the locations given by σ_i: in our example, the state space of $\mathcal{M}_{\sigma_2}$ is

$$\{\underline{3}12\underline{34}1\underline{5}, \underline{3}12\underline{43}1\underline{5}, \underline{4}12\underline{33}1\underline{5}, \ \underline{3}12\underline{35}1\underline{4}, \underline{3}12\underline{45}1\underline{3}, \ \underline{4}12\underline{35}1\underline{3},$$
$$\underline{3}12\underline{53}1\underline{4}, \underline{3}12\underline{54}1\underline{3}, \underline{4}12\underline{53}1\underline{3}, \ \underline{5}12\underline{33}1\underline{4}, \underline{5}12\underline{34}1\underline{3}, \underline{5}12\underline{43}1\underline{3}\}.$$

The moves of $\mathcal{M}_{\sigma_i}$ are a subset of the moves of $\mathcal{M}^*_{\mathrm{T}}$. It rejects all moves of $\mathcal{M}^*_{\mathrm{T}}$ involving an element of $C_1, C_2, \ldots, C_i$. We prove by induction that $\mathcal{M}_{\sigma_i}$ has spectral gap satisfying

$$\mathrm{Gap}(P_i) = \Omega\Big(\Big(n^{k-1-i}\prod_{j=i+1}^{k-1}(c_j + \ln n)\Big)^{-1}\Big),$$

for all choices of σ_i (given a fixed i). Since $\mathcal{M}_{\sigma_0} = \mathcal{M}^*_{\mathrm{T}}$, this will prove the first part of Lemma 4. At each step of the induction, we apply decomposition (Theorem 5). The restrictions of each decomposition will be rapidly mixing by induction and the projection chain will be a bounded generalized exclusion process. The base case is $i = k - 2$ and the final decomposition is $i = 0$.

Base Case. We begin with our base case, $i = k - 2$. Let σ_{k-2} be any fixed location of the particles in classes $C_1, \ldots, C_{k-2}$. The Markov chain $\mathcal{M}_{\sigma_{k-2}}$ rejects all moves of $\mathcal{M}^*_{\mathrm{T}}$ unless they exchange a particle in class C_{k-1} with a particle in class C_k. Thus, its moves only involve two types of particles, with all other particles fixed, so we can view $\mathcal{M}_{\sigma_{k-2}}$ as a generalized exclusion process.

Next we show that $\mathcal{M}_{\sigma_{k-2}}$ is bounded. Consider any "adjacent" particles $x \in C_{k-1}$ and $y \in C_k$ (they could be separated by any number of particles in classes $C_1, \ldots, C_{k-2}$). We select x and the appropriate direction (either L or R) with probability $1/(3n)$. This succeeds with probability $1/2$ if the direction is L. If the direction is R, it succeeds with probability $p_{k,k-1}/(2p_{k-1,k})$ if there are no additional particles between x and y. If there are additional particles, then the probability is even smaller since we are exchanging across elements in smaller classes and our probabilities are weakly monotonic. For example, moving from 3124315 to 4123315 happens with probability $\frac{p_{4,3}p_{4,1}p_{4,2}p_{1,3}p_{2,3}}{2p_{3,4}p_{1,4}p_{2,4}p_{3,1}p_{3,2}} \leq \frac{p_{4,3}}{2p_{3,4}}$. Since for $i < j, C(i) \neq C(j), p_{i,j} > 1/2$, the minimum bias of our generalized exclusion process $\mathcal{M}_{\sigma_{k-2}}$ satisfies $\lambda_L = 1/(p_{k,k-1}/p_{k-1,k}) = p_{k-1,k}/(1 - p_{k-1,k}) > 1$. Hence $\mathcal{M}_{\sigma_{k-2}}$ is bounded, so we can apply Theorem 2. We have c_{k-1} particles in class C_{k-1} and c_k particles in class C_k, and the moves of our exclusion process happen with probability $1/(6n)$ (instead of $1/(c_{k-1} + c_k)$). Thus, Theorem 2 (with $\epsilon = 1/4$) implies that for any such σ_{k-2}, $\mathcal{M}_{\sigma_{k-2}}$ has mixing time $O(n \cdot \min(c_{k-1} + \ln c_k, c_k + \ln c_{k-1}) = O(n(c_{k-1} + \ln n))$. Using Theorem 1(a) we have that the spectral gap of $\mathcal{M}_{\sigma_{k-2}}$ satisfies $\Omega(1/(n(c_{k-1} + \ln n))$.

Inductive Step. We assume by induction the mixing time bound holds for all $\mathcal{M}_{\sigma_i}$ for some $i \leq k-2$, and we will use this result to prove that our mixing time bound holds for all $\mathcal{M}_{\sigma_{i-1}}$, which fix the location of particles in one fewer particle class. Let σ_{i-1} represent any fixed choice of locations for all elements in classes $C_1, C_2, \ldots, C_{i-1}$. In order to bound $\mathrm{Gap}(P_{i-1})$ we use the decomposition theorem. Given any σ_i that is consistent with σ_{i-1} (i.e. they agree on the locations of all elements in classes $C_1, C_2, \ldots, C_{i-1}$), the Markov chain $\mathcal{M}_{\sigma_i}$ will be a restriction Markov chain of $\mathcal{M}_{\sigma_{i-1}}$, as defined in the decomposition theorem. By induction, we have Gap(P_i) satisfies $\text{Gap}(P_i) = \Omega\Big(\Big(n^{k-1-i}\prod_{j=i+1}^{k-1}(c_j + \ln n)\Big)^{-1}\Big)$.

The projection chain, however, is more complicated. Recall our running example ($k = 5$) and consider the second decomposition. Here each of the restrictions is the set of configurations consistent with a particular fixed location of the particles in classes C_1 and C_2 and all restrictions agree on the location of particles in C_1. Let $\sigma_2 = \_112\_\_\_$ and $\beta_2 = 211\_\_\_\_$ represent two such restrictions. A move of the projection chain between σ_2 and β_2 is an aggregate of all moves of $\mathcal{M}^*_{\text{T}}$ between configurations consistent with σ_2 and configurations consistent with β_2. For example, $4112335 \rightarrow 2114335$ is one such move of $\mathcal{M}^*_{\text{T}}$. Each of these moves involve exchanging a particle in C_3 or C_4 with a particle in C_2. However, since these exchanges may happen across any number of particles in C_1 and involve particles in C_3 or C_4 they will have different probabilities, making the analysis more challenging.

More generally, moves of the projection chain involve exchanging an element from C_i with an element from C_j where $j > i$. There may be additional elements between the elements being exchanged but if there are, they are in a smaller particle class C_s with $s < i$. If we view all elements in C_i as one type and all elements in $C_{i+1}, C_{i+2}, \ldots, C_k$ as another, then the projection chain can be viewed as a bounded generalized exclusion process. We show that because of weak monotonicity, all moves that move a particle in C_i ahead happen with probability $1/(6n)$ and all moves that move it back happen with probability at most $(1/(6n))(p_{i+1,i}/p_{i,i+1})$. We defer the proof to the final version of this paper. Since $p_{i,i+1} = 1 - p_{i+1,i} > 1/2$, this implies that the minimum bias is greater than 1 and we can apply Theorem 2. There are c_i particles of type i, $\sum_{j=i+1}^{k} c_j < n$ particles of the other type, and the moves are selected with probability $1/(6n)$. Applying Theorem 2 and Theorem 1(a) shows that the spectral gap of the projection chain satisfies $\Omega((n(c_i + \ln n))^{-1})$. Combining this with the bound on the restriction chain, Theorem 5 implies

$$\text{Gap}(P_{i-1}) = \Omega\Big(\Big(n^{k-i}\prod_{j=i}^{k-1}(c_j + \ln n)\Big)^{-1}\Big). \tag{1}$$

Finally, we will bound the mixing time $\tau_{\text{T}^*}(\epsilon)$ of $\mathcal{M}^*_{\text{T}}$ for $\epsilon > 0$. Let $\lambda_* = \max_{i<j} p_{i,j}/p_{j,i}$ then $\pi_* = \min_{x\in\Omega}\pi(x) \geq (\lambda_*^{\binom{n}{2}} n!)^{-1}$ (see [6] for more details), so $\log(1/\epsilon\pi_*) = O(n^2 \ln \epsilon^{-1})$ since λ is bounded from above by a positive constant. Applying Theorem 1(b) and (1), we have $\tau_{\text{T}^*}(\epsilon) \leq O(n^{2(k-1)} n^2 \ln \epsilon^{-1})$.

4.2 From *k* Particle Process to *k*-Class

Again, we can view $\mathcal{M}_{\text{T}}$ as a product of $k + 1$ smaller Markov chains where ${\mathcal{M}_{\text{T}}}^{(1)}, {\mathcal{M}_{\text{T}}}^{(2)}, \ldots, {\mathcal{M}_{\text{T}}}^{(k)}$ are unbiased nearest-neighbor chains over permutations of a single particle class (moves between elements in the same particle class) and $\mathcal{M}_{\mathrm{T}}^*$ is a k-particle process (moves between elements in different particle classes). We will use the following result of Wilson to bound the mixing times of the k permutation processes and Lemma 4 to bound the mixing time of $\mathcal{M}_{\mathrm{T}}^*$.

Theorem 6 **([5]).** *The chain $\mathcal{M}_{nn}$ mixes in time $O(n^3 \log n \log \epsilon^{-1})$ when $p_{ij} = 1/2$ for all $i < j$.*

Let τ_i be the mixing time of ${\mathcal{M}_{\mathrm{T}}}^{(i)}$ for $1 \leq i \leq k$ and τ_{T^*} be the mixing time of $\mathcal{M}_{\mathrm{T}}^*$. Each chain ${\mathcal{M}_{\mathrm{T}}}^{(i)}$ has c_i particles, so Theorem 6 implies $\tau_i(\epsilon) = O(c_i^3 \log c_i \log \epsilon^{-1})$. By Lemma 4, $\mathcal{M}_{\mathrm{T}}^*$ has mixing time $O(n^{2k} \ln \epsilon^{-1})$. To bound the mixing time of $\mathcal{M}_{\mathrm{T}}$, we will use the following theorem due to Bhakta et al. [6], which bounds the mixing time of a product of independent Markov chains.

Theorem 7 **([6]).** *Suppose the Markov chain $\mathcal{M}$ is a product of N independent Markov chains $\{\mathcal{M}_i\}$, which updates $\mathcal{M}_i$ with probability p_i. If $\tau_i(\epsilon)$ is the mixing time for $\mathcal{M}_i$ and $\tau_i(\epsilon) \geq 4 \ln \epsilon$ for each i, then $\tau(\epsilon) \leq \max_{i=1,2,\ldots,N} \frac{2}{p_i} \tau_i \left(\frac{\epsilon}{2N}\right)$.*

The chain $\mathcal{M}_{\mathrm{T}}$ updates ${\mathcal{M}_{\mathrm{T}}}^{(i)}$ for $1 \leq i \leq k$ if direction N and a particle in class $\mathcal{P}_i$ are selected, which happens with probability $c_i/(6n)$. Therefore, for $i \leq k$,

$$\frac{2}{p_i} \tau_i \left(\frac{\epsilon}{2(k+1)} \right) = O(n c_i^2 \ln c_i \ln(k\epsilon^{-1})) = O(n^3 \ln n \ln(k\epsilon^{-1})).$$

The chain $\mathcal{M}_{\mathrm{T}}^*$ is updated when direction L or R is selected (i.e. with probability $2/3$), so we have $\frac{2}{p_{k+1}} \tau_{\mathrm{T}^*}\left(\epsilon/(2(k+1))\right) = O(n^{2k} \ln(k\epsilon^{-1}))$. Therefore, Theorem 7 gives that $\tau_{\mathrm{T}}(\epsilon) = O(n^{2k} \ln(k/\epsilon))$ for $k \geq 2$. This proves Theorem 3.

5 $\mathcal{M}_{\mathrm{nn}}$ Mixes Rapidly for k-Class Permutations

In the final part of our argument we will use the comparison method [24,25] to bound the mixing time τ_{nn} of $\mathcal{M}_{\mathrm{nn}}$ using the bound on the mixing time of $\mathcal{M}_{\mathrm{T}}$ (Theorem 3). We defer the complete proof to the final version.

Theorem 8. *If the probabilities $\mathcal{P}$ are weakly monotonic and form a bounded k-class for $k \geq 2$, then the mixing time τ_{nn} of $\mathcal{M}_{nn}$ satisfies*

$$\tau_{nn}(\epsilon) = O\left(n^{2k+6} \ln(k\epsilon^{-1})\right).$$

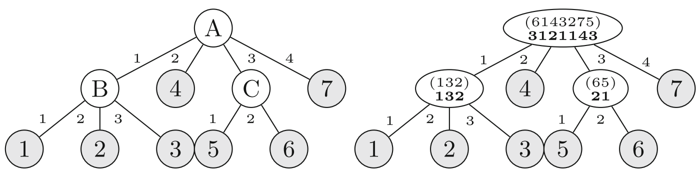
Moreover, as a corollary to Lemma 4, we prove the following result on k-particle systems. In this setting, particles in the same class are indistinguishable. The particle process chain $\mathcal{M}_{\mathrm{pp}}$ is identical to $\mathcal{M}_{\mathrm{nn}}$ except exchanges are only allowed between elements in different classes.

Corollary 9. *If the probabilities $\mathcal{P}$ are weakly monotonic and form a bounded k-class with $k \geq 2$, then the mixing time of $\mathcal{M}_{pp}$ satisfies $\tau_{pp}(\epsilon) = O\left(n^{2k+4} \ln \epsilon^{-1}\right)$.*

6 Trees of k-Value Permutations Mix Rapidly

Bhakta et al. [6] define a class of probabilities they call "League Hierarchies" and show that this class mixes rapidly. A set of probabilities $\mathcal{P}$ is in this class if there exists a binary tree T with n leaves labeled $1, \ldots, n$ in sorted order where each non-leaf node v has a value $\frac{1}{2} \leq q_v < 1$ associated with it and $p_{i,j} = q_{i \wedge j}$ where $i \wedge j$ is the lowest common ancestor of the leaves labeled i and j in T. At a high-level, they show that if the probabilities $\mathcal{P}$ have this type of structure, then we can view the chain as a collection of independent biased 2-particle exclusion processes. We extend their league hierarchies beyond binary trees to allow tree nodes with up to a constant k children.

Let T be a labeled ordered tree (or plane tree) with n leaves labeled $1, \ldots, n$ in sorted order. For each internal node v, the children of v are labeled $1, 2, \ldots, deg(v)$, and if $deg(v) \geq 2$, v is assigned $\binom{deg(v)}{2}$ values $1/2 < q_{(v,a,b)} < 1$, for $1 \leq a < b \leq deg(v)$ (here a and b correspond to the labels of the children of v). Again let $i \wedge j$ be the lowest common ancestor of the leaves labeled i and j. We say that a set of probabilities $\mathcal{P}$ has *k-league structure* if there exists such a tree T with $p_{i,j} = q_{(i \wedge j, a, b)}$ where a and b are children of $i \wedge j$, i is contained in the subtree rooted at a and j is contained in the subtree rooted at b. Figure 2 gives an example $\mathcal{P}$ with k-league structure. In this example, $p_{2,6} = q_{(A,1,3)} = .7$ since $2 \wedge 6 = A$ and leaves 2 and 6 come from A's subtrees labeled 1 and 3 respectively. We prove the chain $\mathcal{M}_{\text{nn}}$ is rapidly mixing for any $\mathcal{P}$ with k-league structure.



$q_{(A,1,2)} = .6$
$q_{(A,1,3)} = .7$
$q_{(A,1,4)} = .8$
$q_{(A,2,3)} = .8$
$q_{(A,2,4)} = .8$
$q_{(A,3,4)} = .7$
$q_{(B,1,2)} = .6$
$q_{(B,1,3)} = .8$
$q_{(B,2,3)} = .8$
$q_{(C,1,2)} = .9$

Fig. 2. A set of probabilities $\mathcal{P}$ with k-league structure, the tree representation of the permutation $\sigma = 6143275$ and the corresponding q values. Each node x is labelled by the sub-permutation σ_x (in parentheses) of σ restricted to the elements appearing as leaves in the subtree rooted at x. For every $i \in \sigma_x$, the bold number below i indicates which child of x has i in its subtree.

Theorem 10. *Given a set of probabilities $\mathcal{P}$ that is weakly monotonic with bounded k-league structure and corresponding tree T with maximum degree k, the mixing time τ_{nn} of $\mathcal{M}_{nn}(T)$ satisfies $\tau_{nn}(\epsilon) = O\left(n^{2k+11} \ln(n\epsilon^{-1})\right)$.*

In the proof (which will appear in the final version), we first show that the tree chain $\mathcal{M}_{\text{tree}}$, introduced by [6], is rapidly mixing and then use comparison techniques to relate the mixing time of $\mathcal{M}_{\text{tree}}$ to the mixing time of $\mathcal{M}_{\text{nn}}$.

References

1. Knuth, D.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, Boston (1969)
2. Aldous, D.J.: Random walks on finite groups and rapidly mixing Markov chains. Séminaire de probabilités de Strasbourg **17**, 243–297 (1983)
3. Diaconis, P., Shahshahani, M.: Generating a random permutation with random transpositions. Z. Wahrscheinlichkeitstheorie Verw. Gebiete **57**, 159–179 (1981)
4. Diaconis, P., Saloff-Coste, L.: Comparison techniques for random walks on finite groups. Ann. Appl. Probab. **21**, 2131–2156 (1993)
5. Wilson, D.: Mixing times of lozenge tiling and card shuffling Markov chains. Ann. Appl. Probab. **1**, 274–325 (2004)
6. Bhakta, P., Miracle, S., Randall, D., Streib, A.: Mixing times of Markov chains for self-organizing lists and biased permutations. In: Proceedings of the 24th ACM/SIAM Symposium on Discrete Algorithms, SODA 2013, pp. 1–15 (2013)
7. Fill, J.: An interesting spectral gap problem (2003, unpublished manuscript)
8. Fill, J.: Background on the gap problem (2003, unpublished manuscript)
9. Benjamini, I., Berger, N., Hoffman, C., Mossel, E.: Mixing times of the biased card shuffling and the asymmetric exclusion process. Trans. Am. Math. Soc. **357**, 3013–3029 (2005)
10. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. Comput. Surv. **17**, 295–311 (1985)
11. Haddadan, S., Winkler, P.: Mixing of permutations by biased transposition. In: 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, pp. 41:1–41:13 (2017)
12. Spitzer, F.: Interaction of Markov processes. Adv. Math. **5**, 240–290 (1970)
13. Luby, M., Randall, D., Sinclair, A.: Markov chain algorithms for planar lattice structures. SIAM J. Comput. **31**, 167–192 (2001)
14. Greenberg, S., Pascoe, A., Randall, D.: Sampling biased lattice configurations using exponential metrics. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009 (2009)
15. Pascoe, A., Randall, D.: Self-assembly and convergence rates of heterogenous reversible growth processes. In: Foundations of Nanoscience (2009)
16. Labbé, C., Lacoin, H.: Cutoff phenomenon for the asymmetric simple exclusion process and the biased card shuffling. arXiv:1610.07383v1 (2016)
17. Levin, D., Peres, Y.: Mixing of the exclusion process with small bias. J. Stat. Phys. **165**, 1036–1050 (2016)
18. Greenberg, S., Randall, D., Streib, A.: Sampling biased monotonic surfaces using exponential metrics (2017)
19. Martin, R., Randall, D.: Sampling adsorbing staircase walks using a new Markov chain decomposition method. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, pp. 492–502 (2000)
20. Martin, R., Randall, D.: Disjoint decomposition of Markov chains and sampling circuits in cayley graphs. Comb. Probab. Comput. **15**, 411–448 (2006)

21. Sinclair, A.: Algorithms for Random Generation and Counting. Progress in Theoretical Computer Science. Birkhäuser, Basel (1993)
22. Randall, D.: Mixing. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (2003)
23. Madras, N., Randall, D.: Markov chain decomposition for convergence rate analysis. Ann. Appl. Probab. **12**, 581–606 (2002)
24. Diaconis, P., Saloff-Coste, L.: Comparison theorems for reversible Markov chains. Ann. Appl. Probab. **3**, 696–730 (1993)
25. Randall, D., Tetali, P.: Analyzing Glauber dynamics by comparison of Markov chains. J. Math. Phys. **41**, 1598–1615 (2000)

Transition Operations over Plane Trees

Torrie L. Nichols[1], Alexander Pilz[2](✉), Csaba D. Tóth[1], and Ahad N. Zehmakan[2]

[1] California State University Northridge, Los Angeles, CA, USA
torrie.nichols.643@my.csun.edu, csaba.toth@csun.edu
[2] Department of Computer Science, ETH Zürich, Zürich, Switzerland
{alexander.pilz,abdolahad.noori}@inf.ethz.ch

Abstract. The operation of transforming one spanning tree into another by replacing an edge has been considered widely, both for general and geometric graphs. For the latter, several variants have been studied (e.g., *edge slides* and *edge rotations*). In a transition graph on the set $\mathcal{T}(S)$ of noncrossing straight-line spanning trees on a finite point set S in the plane, two spanning trees are connected by an edge if one can be transformed into the other by such an operation. We study bounds on the diameter of these graphs, and consider the various operations both on general point sets and sets in convex position. In addition, we address the problem variant where operations may be performed simultaneously. We prove new lower and upper bounds for the diameters of the corresponding transition graphs and pose open problems.

1 Introduction

For a set S of n points in the plane, let $\mathcal{T}(S)$ denote the set of noncrossing straight-line spanning trees on the vertex set S. Over the last 20 years, five operations have been introduced over $\mathcal{T}(S)$. While all five operations are based on a classic exchange property of graphic matroids [27], geometric conditions yield a rich hierarchy.

Elementary Operations. Let $T_1 = (S, E_1)$ and $T_2 = (S, E_2)$ be two trees in $\mathcal{T}(S)$. The operation that replaces T_1 by T_2 is

- an **exchange** if there are edges e_1 and e_2 such that $E_1 \setminus E_2 = \{e_1\}$ and $E_2 \setminus E_1 = \{e_2\}$ (i.e., delete an edge e_1 from E_1 and insert a new edge e_2).
- A **compatible exchange** is an exchange such that the graph $(S, E_1 \cup E_2)$ is a noncrossing straight-line graph (i.e., e_1 and e_2 do not cross).
- A **rotation** is a compatible exchange such that e_1 and e_2 have a common endpoint $p = e_1 \cap e_2$.
- An **empty-triangle rotation** is a rotation such that the edges of neither T_1 nor T_2 intersect the interior of the triangle $\Delta(pqr)$ formed by the vertices of e_1 and e_2.
- An **edge slide** is an empty-triangle rotation such that $qr \in E_1 \cap E_2$.

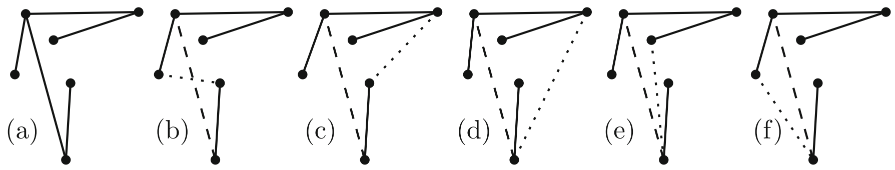


Fig. 1. A straight-line spanning tree (a), in which we replace the dashed edge by a dotted one, using an exchange (b), a compatible exchange (c), a rotation (d), an empty-triangle rotation (e), and an edge slide (f).

See Fig. 1 for illustrations. Note that, for each of these types of operations, the inverse of an operation (i.e., transforming T_2 to T_1) is also of the same type. Each operation **op** defines an undirected *transition graph* $\mathcal{G}_{\mathbf{op}}(S)$, whose vertex set is $\mathcal{T}(S)$, and there is an edge between two trees $T_1, T_2 \in \mathcal{T}(S)$ if an operation **op** can transform T_1 into T_2. The transition graphs for each of these five operations are known to be connected (see Sect. 1.2). The *diameter* $\operatorname{diam}(\mathcal{G}_{\mathbf{op}}(S))$ of the transition graph is thus the maximum length of a shortest transformation sequence between two plane spanning trees in $\mathcal{T}(S)$. We are interested in the asymptotic growth rate of the function $f_{\mathbf{op}}(n) := \max_{|S|=n} \operatorname{diam}(\mathcal{G}_{\mathbf{op}}(S))$.

Simultaneous Operations. For each elementary operation **op**, we define a *simultaneous operation* **sop** on $\mathcal{T}(S)$ as follows. For two trees $T_1 = (S, E_1)$ and $T_2 = (S, E_2)$ in $\mathcal{T}(S)$, the operation **sop** replaces T_1 by T_2 if there is a bijection between $E_1 \setminus E_2$ and $E_2 \setminus E_1$ (the *old* edges and *new* edges, resp.) and each pair (e_1, e_2) of corresponding edges satisfies the condition of the elementary operation **op** on T_1. This means that there is no condition for **simultaneous exchange**; and the only condition for **simultaneous compatible exchange** is that the graph $(S, E_1 \cup E_2)$ is noncrossing (since each of the single operations is valid on T_1, none of the new edges can cross any old edge). We define the graph $\mathcal{G}_{\mathbf{sop}}(S)$ and maximum diameter $f_{\mathbf{sop}}(n)$ for simultaneous operations analogously. Clearly, $f_{\mathbf{sop}}(n) \leq f_{\mathbf{op}}(n)$.

General Position and Convex Position. We assume that S is in *general position* (i.e., no three points in S are collinear). This assumption is for convenience only (all diameter bounds would hold regardless but would require a detailed discussion of special cases). Previous results (cf. Sect. 1.2) are also stated subject to this assumption. Arguably the most important special case is that S is in convex position. We are also interested in the asymptotic growth rate of the function $f^{\text{cx}}_{\mathbf{op}}(n)$, which is equal to $\max_{|S|=n} \operatorname{diam}(\mathcal{G}_{\mathbf{op}}(S))$, where S is in convex position. (Observe that, for the operations mentioned, the actual coordinates of the points are not important as long as they are in convex position.) The function $f^{\text{cx}}_{\mathbf{sop}}(n)$ is defined analogously. Trivially, $f^{\text{cx}}_{\mathbf{op}}(n) \leq f_{\mathbf{op}}(n)$ and $f^{\text{cx}}_{\mathbf{sop}}(n) \leq f_{\mathbf{sop}}(n)$ for any operation **op**.

1.1 Overview of Bounds

The current best diameter bounds for the five operations and their simultaneous variants are summarized in Table 1. Bounds for points in convex position are shown in Table 2. The operations are presented from strongest to weakest: we say an operation $\mathbf{op}_1$ is *stronger* than operation $\mathbf{op}_2$ if every operation $\mathbf{op}_2$ is an operation $\mathbf{op}_1$. As $\mathcal{G}_{\mathbf{op}_2}(S)$ is a subgraph of $\mathcal{G}_{\mathbf{op}_1}(S)$, $f_{\mathbf{op}_1}(n) \leq f_{\mathbf{op}_2}(n)$ and $f^{\mathrm{cx}}_{\mathbf{op}_1}(n) \leq f^{\mathrm{cx}}_{\mathbf{op}_2}(n)$. It is worth noting that even though we briefly review the current best bounds for the three strongest operations, our main results concern the two weakest operations: empty-triangle rotation (Sect. 2) and edge slide (Sect. 3). See Tables 1 and 2, where our contributions are marked with the according theorems and observations.

Table 1. Bounds for n points in general position.

Operation	Single Oper. Upper Bd.	Single Oper. Lower Bd.	Simultaneous Upper Bd.	Simultaneous Lower Bd.
Exchange	$2n-4$	$\lfloor \frac{3n}{2} \rfloor - 5$ [19]	1	1
Compatible Ex.	$2n-4$	$\lfloor \frac{3n}{2} \rfloor - 5$	$O(\log n)$ [3]	$\Omega(\frac{\log n}{\log\log n})$ [11]
Rotation	$2n-4$ [7]	$\lfloor \frac{3n}{2} \rfloor - 5$	$O(\log n)$ (Theorem 1)	$\Omega(\frac{\log n}{\log\log n})$
Empty-Tri. Rot.	$O(n\log n)$ (Theorem 2)	$\lfloor \frac{3n}{2} \rfloor - 5$	$8n$ (Theorem 3)	$\Omega(\log n)$ (Theorem 4)
Edge Slide	$O(n^2)$ [4]	$\Omega(n^2)$ [4]	$O(n^2)$ [4]	$\Omega(n)$ (Sect. 3.1)

Table 2. Bounds for n points in convex position.

Operation	Single Oper. Upper Bd.	Single Oper. Lower Bd.	Simultaneous Upper Bd.	Simultaneous Lower Bd.
Exchange	$2n-5$	$\lfloor \frac{3n}{2} \rfloor - 5$ [19]	1	1
Compatible Ex.	$2n-5$	$\lfloor \frac{3n}{2} \rfloor - 5$	2	2
Rotation	$2n-5$	$\lfloor \frac{3n}{2} \rfloor - 5$	4	3 (Observation 1)
Empty-Tri. Rot.	$2n-5$	$\lfloor \frac{3n}{2} \rfloor - 5$	4 (Theorem 5)	3
Edge Slide	$2n-5$ (Theorem 6)	$\lfloor \frac{3n}{2} \rfloor - 5$	$O(\log n)$ (Theorem 7)	$\Omega(\log n)$ (Theorem 7)

1.2 Related Previous Work and Contribution

Exchanging edges such that both the initial and the resulting graph belong to the same graph class is a well-studied operation in various contexts; see [9] for a survey. Perhaps the best known operation on trees is the classic *rotation* on *ordered rooted binary trees*, which is equivalent to the associativity rule over n-symbol words, and to edge flips in triangulations of $n+2$ points in convex position. Sleator et al. [30] gave an upper bound of $2n-6$ for the diameter of the transition graph for all $n \geq 11$, which is tight [29].

For abstract trees on n labeled vertices, any spanning tree $T_1 = (S, E_1)$ can be transformed into any other tree $T_2 = (S, E_2)$ using $|E_1 \setminus E_2|$ exchange operations, by the classic exchange property of graphic matroids (see, e.g., [27]). Consequently, the diameter of the transition graph is $n-1$.

There are n^{n-2} abstract spanning trees on n labeled vertices for $n \geq 3$ [13]. In contrast, the number of noncrossing straight-line trees on n points in the plane is in $O(141.07^n)$ [20] and $\Omega(12.54^n)$ [21]. While the transition graph of the exchange operation over $\mathcal{T}(S)$ is a subgraph of the transition graph over abstract labeled trees (it has fewer nodes), this does not imply any relation between the diameters of these transition graphs.

The operations of exchange, rotation, and edge slide on *unlabeled* abstract spanning trees on n vertices were considered by Faudree et al. [15], and Goddard and Swart [18]. They define transition graphs over isomorphism classes, proving upper bounds of $n-3$, $2n-6$, and $2n-6$, respectively, on their diameters. For all three types, a lower bound of $n-3$ is the distance between a path and a star.

Geometric variants, where the vertex set S is a set of points in the plane, were first considered by Avis and Fukuda [7] for the efficient enumeration of all trees in $\mathcal{T}(S)$. Interestingly, the order type of S can be reconstructed from the transition graph of the exchange operation [24]. All five operations that we consider were defined prior to our work (see the respective paragraphs), but this is the first comprehensive study of all five operations.

Akl et al. [5] and Chang and Wu [14] considered the exchange operation over $\mathcal{P}(S)$, the set of noncrossing spanning paths on n points in convex position. They proved that the diameter of the transition graph is $2n-6$ for $n \geq 5$ and $2n-5$ for $n = 3, 4$. Wu et al. [31] use these operations for generating all paths in $\mathcal{P}(S)$ in $O(1)$ amortized time per path. Under weaker operations (e.g., compatible exchange), the transition graph of $\mathcal{P}(S)$ is disconnected. It remains open whether the exchange graph of $\mathcal{P}(S)$ is connected for general point sets S.

Exchange. For $n \geq 4$, n points in convex position admit two edge-disjoint spanning trees in $\mathcal{T}(S)$. Since an elementary operation replaces only one edge, this yields a trivial lower bound of $n-1$ for the diameter of the transition graph. Hernando et al. [19] gave a lower bound of $\lfloor 3n/2 \rfloor - 5$ for n points in convex position. An upper bound of $2n-4$ for the general case comes from the weaker operation of rotation. In the simultaneous setting, the lower and upper bound of 1 is clear: Given two trees $T_1, T_2 \in \mathcal{T}(S)$, one can remove all edges of T_1 and insert all edges of T_2 simultaneously.

Compatible Exchange. For single operations, upper and lower bounds follow from corresponding bounds for weaker and stronger operations, respectively. However, the simultaneous compatible exchange is asymptotically weaker than the simultaneous exchange. Buchin et al. [11] constructed a set S of n points and a pair of trees $T_1, T_2 \in \mathcal{T}(S)$ such that $\Omega(\log n/\log\log n)$ simultaneous compatible exchanges are required to transform T_1 into T_2. Aichholzer et al. [3] proved that, for every set S of n points, every $T \in \mathcal{T}(S)$ can be transformed into the minimum spanning tree of S using $O(\log n)$ simultaneous compatible exchanges; moreover, each operation decreases the Euclidean weight of the tree. Later, Aichholzer et al. [2] showed that every $T \in \mathcal{T}(S)$ can be transformed into some canonical tree using $O(\log k)$ simultaneous compatible exchanges, where $k \leq \frac{n}{3}$ is the number of convex layers of S. Their upper bound leaves only a sub-logarithmic gap on the asymptotic growth of $f_{\mathbf{sce}}(n)$, where **sce** stands for simultaneous compatible exchange. However, $f^{\mathrm{cx}}_{\mathbf{sce}}(n) = 2$ because a plane spanning tree T_1 can be transformed into any other plane spanning tree T_2 by

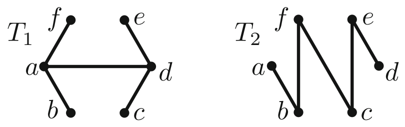


Fig. 2. Two spanning trees on 6 points in convex position. Three simultaneous rotations are needed to transform T_1 into T_2. After the first rotation, ad or its image still crosses some edge of T_2. Thus, at least two more operations are needed to reach T_2.

exchanging all the edges of T_1 with the edges of a path T_0 along the convex hull, and then exchanging all the edges of T_0 with T_2. The existence of two incompatible spanning trees implies a lower bound of 2.

Rotation. The edge rotation operation was first introduced by Chartrand et al. [16] for abstract graphs. We consider it over $\mathcal{T}(S)$. For single rotations **ro**, the lower bound follows from the corresponding bound for stronger operations. An upper bound of $2n - 4$ follows from a proof by Avis and Fukuda [7] (they consider exchange operations, but all exchanges in their proof are happen to be rotations). For simultaneous rotations, **sro**, we prove an $O(\log n)$ upper bound (see Theorem 1, proven in the full version) by an algorithm that transforms every tree $T \in \mathcal{T}(S)$ to a star, combining ideas from [3,7]. A lower bound of $\Omega(\log n/ \log\log n)$ derives from the stronger simultaneous compatible exchanges.

Theorem 1. *$f_{\mathbf{sro}}(n) = O(\log n)$.*

For simultaneous rotations and convex position, an algorithm for the weaker operation of empty-triangle rotations yields an upper bound of 4, and the lower bound is established in Fig. 2 for $n = 6$. For $n \geq 7$, we may augment T_1 and T_2 with $n - 6$ vertices between a and b, and the same argument applies.

Observation 1. *For $n \geq 6$, we have $f^{\text{cx}}_{\mathbf{sro}}(n) \geq 3$.*

Empty-Triangle Rotation. Empty-triangle rotation is a very natural variant of rotation; however, there is not much known about it. Cano et al. [12] showed that a sequence of empty-triangle rotations exists that rotates one edge of a plane spanning tree into any other edge that is not in the tree, which implies the connectivity of $\mathcal{G}_{\mathbf{er}}(S)$ for a point set S, where **er** denotes empty-triangle rotation. For single operations, $f_{\mathbf{er}}(n) \geq \lfloor \frac{3n}{2} \rfloor - 5$ and $f^{\text{cx}}_{\mathbf{er}}(n) \geq \lfloor \frac{3n}{2} \rfloor - 5$ are clear as before. For general point sets, we prove an upper bound of $O(n \log n)$ (see Theorem 2). For the convex case, we provide a linear upper bound in the weaker operation of edge slide, which yields $f^{\text{cx}}_{\text{er}}(n) = \Theta(n)$. In the simultaneous setting, we provide a tight bound of $f_{\mathbf{ser}}(n) = \Theta(\log n)$ in Theorems 3 and 4. In the case of convex position we prove $f^{\text{cx}}_{\mathbf{ser}}(n) = \Theta(1)$ (see Theorem 5).

Edge Slide. Aichholzer et al. [3] proved that $\mathcal{G}_{\mathbf{es}}(S)$, where **es** stands for edge slide, is connected for every point set S in general position. Aichholzer and Reinhardt [4] proved $f_{\mathbf{es}}(n) = \Theta(n^2)$. We show $f^{\text{cx}}_{\mathbf{es}}(n) = \Theta(n)$; see Theorem 6.

The simultaneous variant has not been previously considered. A linear lower bound and a quadratic upper bound can be easily derived from diameter bounds for single operations over point sets in general position, as will be discussed in Sect. 3.1. For points in convex position, however, we prove an asymptotically tight bound $f^{\text{cx}}_{\textbf{ses}}(n) = \Theta(\log n)$; see Theorem 7.

2 Empty-Triangle Rotation: Upper and Lower Bound

2.1 General Point Sets

For single operations, the lower bound of $\lfloor \frac{3n}{2} \rfloor - 5$ follows from an analogous bound for the stronger operation of rotation. We prove an upper bound of $O(n \log n)$ (see Theorem 2), which leaves a logarithmic gap. We start with an easy observation about a single triangle.

Observation 2. *Let T be a spanning tree with an edge $e = pq$ and a vertex r such that the interior triangle $\Delta(pqr)$ does not intersect T. Then an empty-triangle rotation can replace e with either pr or qr (but not both).*

Theorem 2. $f_{\textbf{er}}(n) = O(n \log n)$.

Proof. Let T be a spanning tree in $\mathcal{T}(S)$ for a point set S of size n and let $p \in S$ be an extremal point in S. We show that we can transform T into a star centered at p using $O(n \log n)$ empty-triangle rotations. To this end, we use $O(n)$ operations to transform T into two subtrees of roughly equal size whose convex hulls intersect in p only, and then recurse on the subtrees.

Let h be a ray emanating from p that separates the convex hull of S into two parts, none containing more than $n/2$ points of $S \setminus \{p\}$. If h does not cross any edge of T, we can recurse on the two subtrees. Otherwise, let e be the edge of T whose crossing with h is farthest away from p. Compute a triangulation of S whose edge set contains the edge set of T, and let $(\Delta_1, \ldots, \Delta_m)$ be the sequence of triangles of the triangulation, in the order in which they are visited by h, until h crosses e. (Note that $m \leq 2n - 5$, as this is an upper bound for the number of triangles in a triangulation of S.) By Observation 2, an empty-triangle rotation can replace e with some other edge f of Δ_m. This edge f either does not cross h, or its crossing with h is closer to p than the one with e. In any case, we obtain a tree $T' \in \mathcal{T}(S)$ whose edge set is contained in the same triangulation; however, the sequence of triangles visited by h until the last crossing with an edge in T' is now $(\Delta_1, \ldots, \Delta_{m'})$ for some $m' < m$. Consequently, after at most $m \leq 2n - 5$ empty-triangle rotations, h does not cross any edge of the tree, and we can recurse on the two subtrees, each of size at most $n/2 + 1$.

When every subtree contains only two vertices, then all edges are incident to p, and their union is a star centered at p. The number $a(n)$ of operations needed to transform T into a star centered at p satisfies the recurrence relation $a(n) \leq 2a(n/2 + 1) + O(n)$, which solves to $O(n \log n)$. Since any two trees in $\mathcal{T}(S)$ can be transformed into a star centered at p using $a(n)$ operations, we have $f_{\textbf{er}}(n) = O(n \log n)$. □

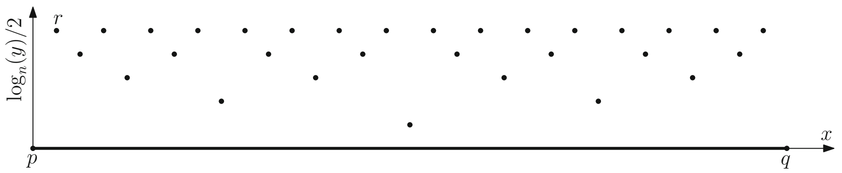


Fig. 3. (a) The schematic image of point set S for $n = 2^5 + 1 = 33$ and $k = 5$, on a logarithmic scale. The edge pq is horizontal and vertex $q = (1, n^{2k}) = (1, 33^{10})$ has maximal y-coordinate. (The logarithmic scale distorts the slopes.)

A simultaneous empty-triangle rotation consists of one or more empty-triangle rotations that can be performed independently. The empty triangles involved in such an operation are interior-disjoint, and at most one edge rotates in every empty triangle. For the simultaneous variant, $f_{\mathbf{ser}}(n)$, we provide a linear upper bound and a logarithmic lower bound.

Theorem 3. $f_{\mathbf{ser}}(n) < 8n$.

Proof. Revisit the proof of Theorem 2. It takes at most $2n - 5$ empty-triangle rotations to split the initial tree into two subtrees, each of size at most $n/2 + 1$. However, the set of triangles involved in the recursive calls are disjoint, and rotations can be done simultaneously. The number of rotations to obtain a star is therefore bounded by the recursion $a(n) \leq 2n - 5 + a(n/2 + 1) < 4n$. ☐

Theorem 4. $f_{\mathbf{ser}}(n) = \Omega(\log n)$.

Proof. We may assume that $n = 2^k + 1$ for some $k \in \mathbb{N}$. We construct a point set S and two spanning trees $T_1, T_2 \in \mathcal{T}(S)$ such that it takes at least $k = \log_2(n-1)$ simultaneous empty-triangle rotations to transform T_1 into T_2. The points in S have integer coordinates, and are not in general position, but a random perturbation by a small $\varepsilon > 0$ would bring S to general position and preserve all combinatorial properties in our proof.

Our point set is $S = \{(x, \varphi(x)) : x = 0, \ldots, n\}$, where we define $\varphi(x) : \{0, \ldots, n\} \to \mathbb{N}_0$ as follows; refer to Fig. 3. Every integer $x \in \{0, 1, \ldots, n\}$ has a binary representation $x = \sum_{i=0}^{k} x_i 2^i$ with $x_i \in \{0, 1\}$. For $x = 1, \ldots, n-1$, let $j(x)$ be the smallest index such that $x_{j(x)} = 1$. For $x = 1, \ldots, n-1$, let $\varphi(x) = n^{2(k-j(x))}$; and let $\varphi(0) = \varphi(n) = n^0 = 1$. This completes the definition of S. Let T_1 and T_2, respectively, be a star centered at $p = (0, 1)$ and $r = (1, \varphi(1)) = (1, n^{2k})$.

Consider a sequence of simultaneous empty-triangle operations that carry T_1 to T_2. Each edge of T_1 has a trajectory (i.e., a sequence starting with that edge in which each edge is followed by its replacement), and is eventually transformed into an edge in T_2 incident to q. We trace the trajectory of the edge $e = pq$, with $q = (n, 1)$ of T_1, and show that it takes at least k operations to transform e into an edge incident to q.

For $i = 0, 1, \ldots, k-1$, denote by S_i the set of points in S whose y-coordinate is at most n^{2i}, that is, $S_i = \{(a, \varphi(a)) \in S : \varphi(a) \leq n^{2i}\}$. We claim that if an empty triangle spanned by S has two vertices in S_i ($i = 0, \ldots, k-1$), then the third vertex must be in S_{i+1}. To prove this claim, we make a few observations about points in S and the slopes of line segments spanned by S.

By construction, for any two points $a, b \in S_i$ ($i = 0, \ldots, k-1$), there is a point $m \in S_{i+1} \setminus S_i$, whose x-coordinate is between that of a and b. The slope of a segment between points $a = (x_a, y_a)$ and $b = (x_b, y_b)$ is defined as $\text{slope}(ab) = (y_a - y_b)/(x_a - x_b)$. In particular, for any two points $a, b \in S_i$, we have $|\text{slope}(ab)| \leq n^{2i}$. For $a \in S_i$ and $b \in S_{i+1} \setminus S_i$, we have $\frac{1}{2}n^{2i+1} < (n^{2i+2} - n^{2i})/n < |\text{slope}(ab)| < n^{2i+2}$. For $a \in S_i$ and $b \in S \setminus S_{i+1}$, we have $\frac{1}{2}n^{2i+3} < (n^{2i+4} - n^{2i})/n < |\text{slope}(ab)|$.

We are now ready to prove our claim. Consider an empty triangle $\Delta(abc)$ with $a, b \in S_i$, $x_a < x_b$, and $c \in S \setminus S_i$. If $c \notin S_{i+1}$, then there exists a point $m \in S_{i+1}$ such that $x_a < x_m < x_b$. As noted above, we have $|\text{slope}(ab)| < |\text{slope}(am)| < |\text{slope}(ac)|$, and $|\text{slope}(ab)| < |\text{slope}(bm)| < |\text{slope}(bc)|$. Consequently, m lies in the interior of $\Delta(abc)$, contrarily to our assumption that this triangle is empty.

It follows that a simultaneous empty-triangle rotation transforms every edge spanned by S_i to an edge spanned by S_{i+1}, for $i = 0, \ldots, k-1$. In particular, the edge $e = pq$ is spanned by S_0, and the point r is in $S_k \setminus S_{k-1}$. Consequently, it takes at least k operations to transform edge e to an edge incident to r. $\square$

2.2 Convex Position

A construction in [19] designed for the stronger exchange operation yields the lower bound $\lfloor \frac{3n}{2} \rfloor - 5$ for single empty-triangle rotations for point sets in convex position. Similarly, we can derive an upper bound of $f^{\text{cx}}_{\textbf{er}}(n) \leq 2n - 5$ from our algorithm for edge slides (Theorem 6). In Theorem 5 below, we provide a constant upper bound for simultaneous empty-triangle rotations.

We define the *dual tree* of a plane tree $T \in \mathcal{T}(S)$ for a set S of $n \geq 3$ points in convex position as follows. The edges of T subdivide the convex n-gon $\text{conv}(S)$ into one or more convex *cells*, which correspond to the nodes of the dual tree. Two nodes of the dual tree are adjacent if the corresponding cells share an edge. Note that the dual tree is indeed a tree. Furthermore, the boundary of each cell contains precisely one edge that is not in T, and this edge is necessarily an edge of $\text{conv}(S)$; we call this edge the *hull edge* of the cell. The main idea of the proof of the following theorem, given in the full version, is to rotate edges shared by cells to hull edges.

Theorem 5. $f^{\text{cx}}_{\textbf{ser}}(n) \leq 4$.

3 Edge Slide

For simultaneous edge slides, we may also consider the following, more restricted variant. Consider a plane spanning tree T on a point set S. Two edge slide operations that move v_1u_1 to v_1w_1 and v_2u_2 to v_2w_2, respectively, can be performed simultaneously if the triangles $\Delta(u_1v_1w_1)$ and $\Delta(u_2v_2w_2)$ intersect in at most one point. All lower bounds in this section hold for the more powerful setting (in which the edge along we slide can be shared), and the upper bounds apply to the more restricted setting that does not allow a shared edge.

3.1 General Point Sets

As noted above, Aichholzer and Reinhardt [4] proved that $f_{\mathbf{es}}(n) = \Theta(n^2)$. Little is known about the simultaneous variant. However, their results immediately imply $f_{\mathbf{ses}}(n) = O(n^2)$. A lower bound of $f_{\mathbf{ses}}(n) = \Omega(n)$ can also be derived easily from $f_{\mathbf{es}}(n) = \Omega(n^2)$, as one simultaneous edge slide is not stronger than $\lfloor \frac{n-1}{2} \rfloor$ sequential edge slides, and at most $\lfloor \frac{n-1}{2} \rfloor$ edges can slide at once (for every pair of edges involved in a slide, only one changes).

3.2 Convex Position

For single edge slide operations, the lower bound $\lfloor \frac{3n}{2} \rfloor - 5$ is trivial, and Theorem 6, building on Lemma 1 below (which is proven in the full version), provides a linear upper bound.

Lemma 1. *Given a set S of $n \geq 3$ points in convex position and two paths P_1 and P_2, both of which are paths along edges of* $\operatorname{conv}(S)$*, we can transform P_1 to P_2 using $n-2$ edge slides.*

Theorem 6. $f^{\mathrm{cx}}_{\mathbf{es}}(n) \leq 2n - 5$.

Proof. We show that any two plane spanning trees $T_1, T_2 \in \mathcal{T}(S)$ on a set S of $n \geq 3$ points in convex position can be transformed into a path P with a sequence of at most $n-3$ and $n-2$ edge slides, respectively.

Consider the dual tree of T_1 and choose a leaf C_0 of the dual tree to be the root. Denote by e_0 the hull edge of C_0 (i.e., the edge of C_0 that is not in T_1); and let $P \in \mathcal{T}(S)$ be the path formed by the remaining $n-1$ edges of $\operatorname{conv}(S)$. If the dual tree has only one node, then $T_1 = P$. Otherwise, let C_1 be a child of C_0 in the dual tree and let e_1 be the edge shared by C_0 and C_1. Since C_1 is the convex hull of its vertices, we can apply Lemma 1 to slide e_1 to the hull edge of cell C_1. As a result, cells C_0 and C_1 are merged to one cell. We let this cell be the new root cell and iterate. Each edge slide increases the size of the root cell by one, so we reach P after at most $n-3$ edge slides.

If edge e_0 is absent from T_2, we can transform T_2 to P as described above using $n-3$ edge slides. However, if e_0 is an edge of T_2, we apply an edge slide to replace e_0 with some other edge, followed by a sequence of $n-3$ edge slides to obtain P. The total number of operations is at most $2n-5$, as claimed. □

Now, let us consider simultaneous edge slides. In the proof of the following result, we repeatedly apply a reduction step that "removes" a constant fraction of the leaves; this idea was originally developed for simultaneous flip operations in triangulations [8,17].

Theorem 7. $f^{\mathrm{cx}}_{\mathbf{ses}}(n) = \Theta(\log n)$.

Proof. For the lower bound, consider two different paths P_1 and P_2 along the convex hull of a point set of size n in convex position where the edge uv is in P_2 but not in P_1. Since P_1 is a path along the convex hull, there is only one cell C_0, which is incident to all n vertices. To transform P_1 into P_2, one has to slide edges of P_1 until C_0 vanishes (i.e., its size drops to 2). The size of C_0 decreases only if an edge of C_0 slides along another edge of C_0, that is, any size-decreasing edge slide involves two consecutive edges of C_0. Consequently, a simultaneous edge slide decreases the size of C_0 by at most a factor of 2; and so any sequence of simultaneous edge slides must use at least $\log_2(n/2) = \Omega(\log n)$ operations.

For the upper bound, let S be a set of $n \geq 3$ points in convex position, let $p \in S$ and $T_1 \in \mathcal{T}(S)$. It is sufficient to show that T_1 can be transformed to a star centered at p using $O(\log n)$ simultaneous edge slides.

The outline of the proof is as follows. We transform T_1 into a star centered at p via some intermediate phases where each phase uses $O(\log n)$ simultaneous edge slide operations. The assumption that S is in convex position is crucial for maintaining the planarity of the intermediate trees.

First, we show how to transform T_1 into a spanning tree T_2 in which every cell has $O(1)$ edges. Then, we transform T_2 into a tree T_3 of height $O(\log n)$. Finally, T_3 is transformed into a star centered at p. All these steps require $O(\log n)$ simultaneous edge slides.

Constant-size cells. Pick an arbitrary convex hull edge pq and define the cell incident to pq to be the root of the dual tree. The edge that separates a cell from its parent is called the *parent edge*. The hull edge and the parent edge split the boundary of a cell into two paths. Note that these paths are convex. A simultaneous edge slide can decrease a convex path of $k \geq 2$ edges to a chain of $\lceil k/2 \rceil$ edges by sliding every other edge along the previous edge along the path. Note that these slides can be performed simultaneously in a cell C; if an edge e involved in a slide is incident to another cell C', then e is the parent edge of C' and therefore there is no slide in C' that involves e. If the same operation is performed on the parent cell, one new edge may also be inserted into the cell. After $O(\log n)$ simultaneous edge slides (each modifying all convex paths of length two or higher), we obtain a tree T_2 whose cells each have at most six edges. (If a cell has seven or more edges, then the two paths loose two or more edges, and the cell may gain only one edge from its parent.) Let n_c be the number of nodes of the dual tree of T_2. The number of incident cell-edge pairs over all edges of $\mathrm{conv}(S)$ and T_2 is $n + 2(n_c - 1)$ and this number is at most $6n_c$. Therefore, $n_c \geq (n-2)/4$.

Creating good leaves. A leaf of a spanning tree T (or subtree) is called *good* if the edge incident to it is an edge of the convex hull of the vertices of T. Note that if we remove a good leaf from T to obtain a tree T', then edge slides on the resulting tree T' can also be performed in the entire tree T (that is, the edge of a good leaf does not obstruct any edge slide in T'). The main idea of transforming T_2 to a tree T_3 of height $O(\log n)$ is to repeatedly create and "remove"

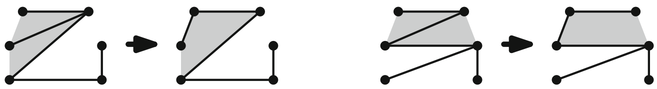


Fig. 4. Small cells of degree 2 are merged by one slide only involving the parent edge.

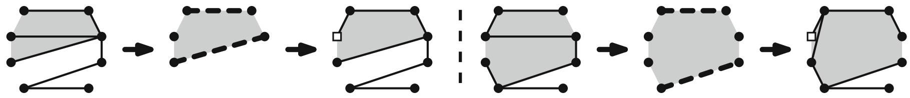


Fig. 5. For cells with a single child and a single grandchild such that the cell and its child have at least five vertices, we can perform a constant number of edge slides to obtain a good leaf resulting in one (left) or two (right) new cells.

a constant fraction of good leaves (meaning that these leaves are disregarded in later iterations).

Let T_2 be a spanning tree with cells of size at most six. Let $n_1+n_2+n_3 = n_c$ denote the number of nodes of the dual tree of degree 1, 2, and more, respectively. Note that $n_1 \geq n_3$. We show that we can always remove a constant fraction of the good leaves of T_2 by considering the leaves and the degree 2 vertices of the dual tree.

First we perform the following "clean-up." Consider first the nodes with a single child in the dual tree that are at even distance from the root. If the corresponding cell and the cell of its child jointly have at most four vertices, we transform them to a single cell with a constant number of slides (see Fig. 4). Then, we do the analogous transformation for the cells at odd distance from the root in the dual tree (if they were not already altered by the previous process).

After this process, consider any cell C with a single child C' and a single grandchild C''. Together, the cells C and C' have at least five vertices. The parent edge of C and the parent edge of C'' are incident to at most four vertices. Hence, we can slide the edges of C and C' such that there is a fifth vertex that becomes a good leaf. Depending on the number of edges incident to C and C', the cells are transformed to one or two new cells (to maintain the invariant that each cell has at most six edges). See Fig. 5.

Let us count the number of good leaves we can obtain with these operations. For each of the n_1 leaves of the dual tree, we obtain at least one good leaf. Then, we create at least one good leaf for every disjoint pair of adjacent nodes of degree 2 in the dual tree. There can be at most n_1+n_3-1 nodes of degree 2 that cannot be paired up with one of its neighbors. (Every maximal chain of nodes of degree 2 can be considered a subdivision of a single edge in a tree of $n_1 + n_3$ vertices.) From the remaining (paired) nodes, we extract at least $(n_2-n_1-n_3+1)/2$ good leaves. Suppose $n_1 < n_c/6$. Then, as $n_1 \geq n_3$, we have $n_2 > 4n_c/6$ and thus $(n_2 - n_1 - n_3 + 1)/2 \geq n_c/6$. Hence, after transforming the cells of degree 2, we have at least $n_c/6 \geq (n-2)/24$ good leaves (using the bound $n_c \geq (n-2)/4$ obtained above).

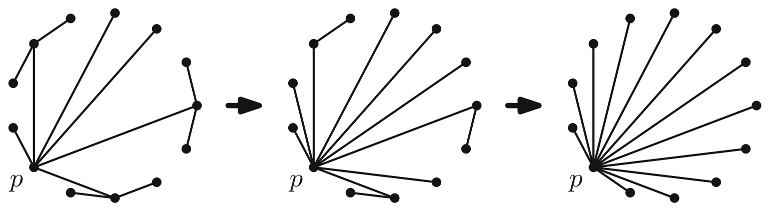


Fig. 6. One round of sliding good leaves of a subtree into a star centered at p.

We can now summarize the steps to transform T_2 to T_3 using $O(\log n)$ simultaneous edge slide operations. Starting from T_2, we repeatedly create good leaves and remove them. Denote by L_i the set of good leaves removed in iteration i. Each iteration removes at least a $\frac{1}{24}$-fraction of the vertices. After $r \in O(\log n)$ iterations, we are left with a single vertex p. The tree $T_3 \in \mathcal{T}(S)$ is the tree obtained by these "removal" operations. In each iteration, the edges incident to good leaves are edges of the convex hull of the current subtree, consequently at most two such edges are incident to the same vertex in the subtree.

Creating a star. The one-vertex tree on p is a star. We re-insert the leaves in L_i for $i = r, r-1, \ldots, 1$ (in reverse order) in r rounds, and transform the subtree into a star centered at p. In round j, we re-add the edges to the vertices in L_{r+1-j}. They are each adjacent to the current star centered at p, and each vertex of the current star is incident to at most two edges in L_{r+1-j}. Using two simultaneous edge slides, all the edges in L_{r+1-j} become incident to p. See Fig. 6. After r rounds, we obtain the star centered at p. As noted above, edge slides performed in a subtree can also be performed in the whole tree, as the edges incident to good leaves do not obstruct any edge slides. This completes the proof. □

4 Conclusions

Previous work introduced five elementary operations on the space of plane spanning trees $\mathcal{T}(S)$ on a point set S in Euclidean space. All five operations are known to define a connected transition graph. This is the first comprehensive analysis of the diameters of these graphs. Obvious open problems are to close the gaps between the lower and upper bounds in Tables 1 and 2. One might also consider new variations. For example, we obtain a new variant of *empty-triangle rotation* if we require $\Delta(pqr)$ to be empty of vertices (but not necessarily edges), or a new variant of *edge slide* when not requiring $\Delta(pqr)$ to be empty. These variations have not been considered and may lead to new geometric insight.

Transition graphs on other common plane geometric graphs have been considered in the literature, but they do not allow for such a rich variety of operations. For the space of noncrossing matchings on S, a compatible exchange operation has been defined, but the transition graph is disconnected even if S is in convex position [1]; it is known that the transition graph has no isolated vertices [22]. Connectivity is known for bipartite geometric matchings, with a tight linear diameter bound [6]. For noncrossing Hamiltonian cycles (a.k.a. polygonizations) it is a longstanding open problem whether the transition graph of simultaneous compatible exchange is connected.

Transition graphs of edge flips in geometric triangulations of S have been studied extensively. Recently, Bose et al. [10] considered the orbits of individual edges. Lubiw et al. [25] prove that a flip sequence of length $O(n^7)$ can carry any edge-labeled triangulation to any other (by showing that the 2-skeleton of the flip complex is contractible). Analogous problems for edge-labeled plane spanning trees (under all five elementary operations) remain open.

While the upper bounds on the transition graph diameter are constructive, the problem of determining the transformation distance between two given trees seems to be still open (or is trivial) in all settings we discussed. Similar problems have been studied for triangulations: it is NP-hard to determine the flip distance of two triangulations of a point set [26,28], but the problem is fixed-parameter tractable in their distance [23]. Even though the transition graph has a small diameter for simultaneous operations, the degree may be exponential, and the distance between two trees thus does not seem to be a suitable parameter for the complexity of the problem. For convex position, the complexity of the related problem on triangulations (already posed in [30]) is still open.

Acknowledgment. Key ideas for our results on simultaneous edge slides were discussed at the GWOP 2017 workshop in Pochtenalp, Switzerland. We thank all participants for the constructive atmosphere. Research by Nichols and Tóth was partially supported by the NSF awards CCF-1422311 and CCF-1423615. Pilz is supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35.

References

1. Aichholzer, O., Asinowski, A., Miltzow, T.: Disjoint compatibility graph of non-crossing matchings of points in convex position. Electron. J. Comb. **22**, P1 (2015)
2. Aichholzer, O., Aurenhammer, F., Huemer, C., Krasser, H.: Transforming spanning trees and pseudo-triangulations. Inf. Process. Lett. **97**(1), 19–22 (2006)
3. Aichholzer, O., Aurenhammer, F., Hurtado, F.: Sequences of spanning trees and a fixed tree theorem. Comput. Geom. **21**(1–2), 3–20 (2002)
4. Aichholzer, O., Reinhardt, K.: A quadratic distance bound on sliding between crossing-free spanning trees. Comput. Geom. **37**(3), 155–161 (2007)
5. Akl, S.G., Islam, M.K., Meijer, H.: On planar path transformation. Inf. Process. Lett. **104**(2), 59–64 (2007)
6. Aloupis, G., Barba, L., Langerman, S., Souvaine, D.L.: Bichromatic compatible matchings. Comput. Geom. **48**(8), 622–633 (2015)
7. Avis, D., Fukuda, K.: Reverse search for enumeration. Discret. Appl. Math. **65**(1–3), 21–46 (1996)
8. Bose, P., Czyzowicz, J., Gao, Z., Morin, P., Wood, D.R.: Simultaneous diagonal flips in plane triangulations. J. Graph Theory **54**(4), 307–330 (2007)

9. Bose, P., Hurtado, F.: Flips in planar graphs. Comput. Geom. **42**(1), 60–80 (2009)
10. Bose, P., Lubiw, A., Pathak, V., Verdonschot, S.: Flipping edge-labelled triangulations. Comput. Geom. (2017, in Press)
11. Buchin, K., Razen, A., Uno, T., Wagner, U.: Transforming spanning trees: a lower bound. Comput. Geom. **42**(8), 724–730 (2009)
12. Cano, J., Díaz-Báñez, J.M., Huemer, C., Urrutia, J.: The edge rotation graph. Graphs Comb. **29**(5), 1–13 (2013)
13. Cayley, A.: A theorem on trees. Q. J. Math. **23**, 376–378 (1889)
14. Chang, J.M., Wu, R.Y.: On the diameter of geometric path graphs of points in convex position. Inf. Process. Lett. **109**(8), 409–413 (2009)
15. Faudree, R., Schelp, R., Lesniak, L., Gyárfás, A., Lehel, J.: On the rotation distance of graphs. Discret. Math. **126**(1), 121–135 (1994)
16. Chartrand, G., Saba, F., Zou, H.B.: Edge rotations and distance between graphs. Cas. Pest. Math. **110**, 87–91 (1985)
17. Galtier, J., Hurtado, F., Noy, M., Pérennes, S., Urrutia, J.: Simultaneous edge flipping in triangulations. Int. J. Comput. Geom. Appl. **13**(2), 113–134 (2003)
18. Goddard, W., Swart, H.C.: Distances between graphs under edge operations. Discret. Math. **161**(1), 121–132 (1996)
19. Hernando, M., Hurtado, F., Márquez, A., Mora, M., Noy, M.: Geometric tree graphs of points in convex position. Discret. Appl. Math. **93**(1), 51–66 (1999)
20. Hoffmann, M., Sharir, M., Sheffer, A., Tóth, C.D., Welzl, E.: Counting plane graphs: flippability and its applications. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 524–535. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22300-6_44
21. Huemer, C., de Mier, A.: Lower bounds on the maximum number of non-crossing acyclic graphs. Europ. J. Comb. **48**, 48–62 (2015)
22. Ishaque, M., Souvaine, D.L., Tóth, C.D.: Disjoint compatible geometric matchings. Discret. Comput. Geom. **49**, 89–131 (2013)
23. Kanj, I.A., Sedgwick, E., Xia, G.: Computing the flip distance between triangulations. Discret. Comput. Geom. **58**(2), 313–344 (2017)
24. Keller, C., Perles, M.A.: Reconstruction of the geometric structure of a set of points in the plane from its geometric tree graph. Discret. Comput. Geom. **55**(3), 610–637 (2016)
25. Lubiw, A., Masárová, Z., Wagner, U.: A proof of the orbit conjecture for flipping edge-labelled triangulations. In: Proceedings of the 33rd Symposium on Computational Geometry (SoCG 2017). LIPIcs, vol. 77, pp. 49:1–49:15. Schloss Dagstuhl (2017)
26. Lubiw, A., Pathak, V.: Flip distance between two triangulations of a point set is NP-complete. Comput. Geom. **49**, 17–23 (2015)
27. Oxley, J.G.: Matroid Theory. Oxford University Press, Oxford (1993)
28. Pilz, A.: Flip distance between triangulations of a planar point set is APX-hard. Comput. Geom. **47**(5), 589–604 (2014)
29. Pournin, L.: A combinatorial method to find sharp lower bounds on flip distances. In: Proceedings of the 25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013), pp. 1–12 (2013)
30. Sleator, D.D., Tarjan, R.E., Thurston, W.P.: Rotation distance, triangulations, and hyperbolic geometry. J. Am. Math. Soc. **1**, 647–681 (1988)
31. Wu, R.Y., Chang, J.M., Pai, K.J., Wang, Y.L.: Amortized efficiency of generating planar paths in convex position. Theor. Comput. Sci. **412**(35), 4504–4512 (2011)

Analysis of the Continued Logarithm Algorithm

Pablo Rotondo[1,2,3], Brigitte Vallée[2](✉), and Alfredo Viola[3]

[1] IRIF, CNRS and Université Paris Diderot, Paris, France
rotondo@irif.fr
[2] GREYC, CNRS and Université de Caen, Caen, France
brigitte.vallee@unicaen.fr
[3] Universidad de la República, Montevideo, Uruguay
viola@fing.edu.uy

Abstract. The Continued Logarithm Algorithm –*CL* for short– introduced by Gosper in 1978 computes the gcd of two integers; it seems very efficient, as it only performs shifts and subtractions. Shallit has studied its worst-case complexity in 2016 and showed it to be linear. We here perform the average-case analysis of the algorithm: we study its main parameters (number of iterations, total number of shifts) and obtain precise asymptotics for their mean values. Our "dynamical" analysis involves the dynamical system underlying the algorithm, that produces continued fraction expansions whose quotients are powers of 2. Even though this *CL* system has already been studied by Chan (around 2005), the presence of powers of 2 in the quotients ingrains into the central parameters a dyadic flavour that cannot be grasped solely by studying the *CL* system. We thus introduce a dyadic component and deal with a two-component system. With this new mixed system at hand, we then provide a complete average-case analysis of the *CL* algorithm, with explicit constants (Thanks to the `Dyna3S` ANR Project and the `AleaEnAmsud` AmSud-STIC Project.).

1 Introduction

In an unpublished manuscript [6], Gosper introduced the continued logarithms, a mutation of the classical continued fractions. He writes "The primary advantage is the conveniently small information parcel. The restriction to integers of regular continued fractions makes them unsuitable for very large and very small numbers. The continued fraction for Avogadro's number, for example, cannot even be determined to one term, since its integer part contains 23 digits, only 6 of which are known. (...) By contrast, the continued logarithm of Avogadro's number begins with its binary order of magnitude, and only then begins the description equivalent to the leading digits – a sort of recursive version of scientific notation".

The idea of Gosper gives rise to an algorithm for computing gcd's, described by Shallit in [8]. This algorithm has two advantages: first, it can be calculated starting from the most representative bits, and uses very simple operations (subtractions and shifts); it does not employ divisions. Second, as the quotients which

© Springer International Publishing AG, part of Springer Nature 2018
M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 849–863, 2018.
https://doi.org/10.1007/978-3-319-77404-6_61

intervene in the associated continued fraction are powers of two 2^a, we can store each of them with $\log_2 a$ bits. Then, the algorithm seems to be of small complexity, both in terms of computation and storage.

Shallit [8] performs the worst-case analysis of the algorithm, and studies the number of steps $K(p,q)$, and the total number of shifts $S(p,q)$ that are performed on an integer input (p,q) with $p<q$: he proves the inequalities

$$K(p,q) \leq 2\log_2 q + 2, \quad S(p,q) \leq (2\log_2 q + 2)\,\log_2 q, \tag{1}$$

and exhibits instances, namely the family $(2n-1,1)$, which show that the previous bounds are nearly optimal, $K(1,2n-1) = 2n-2, S(1,2n-1) = n(n-1)/2+1$.

In a personal communication, Shallit proposed us to perform the average-case analysis of the algorithm. In this paper, we answer his question. We consider the set Ω_N which gathers the integer pairs (p,q) with $0 \leq p \leq q \leq N$, endowed with the uniform probability, and we study the mean values $\mathbb{E}_N[K]$ and $\mathbb{E}_N[S]$ as $N \to \infty$. We prove that these mean values are asymptotically linear in the size $\log N$, and exhibit their precise asymptotic behaviour for $N \to \infty$,

$$\mathbb{E}_N[K] \sim \frac{2}{H}\log N, \quad E_N[S] \sim \frac{\log 3 - \log 2}{2\log 2 - \log 3}\mathbb{E}_N[K].$$

The constant H is related to the entropy of an associated dynamical system and

$$H = \frac{1}{2\log 2 - \log 3}\left[\frac{\pi^2}{6} + 2\sum_{k\geq 1}\frac{(-1)^k}{k^2\,2^k} - (\log 2)(3\log 3 - 4\log 2)\right]. \tag{2}$$

This entails numerical estimates (validated by experiments) for the mean values

$$\mathbb{E}_N[K] \sim 1.49283\log N, \quad \mathbb{E}_N[S] \sim 1.40942\log N.$$

Then, from (1), the mean number of divisions is about half the maximum.

Our initial idea was to perform a dynamical analysis along the lines described in [10]. The *CL* algorithm is defined as a succession of steps, each consisting of a pseudo-division which transforms an integer pair into a new one. This transformation may be read first on the associated rationals and gives rise to a mapping T that is further extended to the real unit interval $\mathcal{I}$. This smoothly yields a dynamical system $(\mathcal{I}, T)$, the *CL* system, already well studied, particularly by Chan [3] and Borwein [1]. The system has an invariant density ψ (with an explicit expression described in (8)) and is ergodic. Thus we expected this dynamic analysis to follow general principles described in [10].

However, the analysis of the algorithm is not so straightforward. The binary shifts, which make the algorithm so efficient, cause many problems in the analysis. Even on a pair of coprime integers (p,q), the algorithm creates intermediate pairs (q_{i+1}, q_i) which are no longer generally coprime, as their gcd is a non-trivial power of 2. These extra gcd's are central in the analysis of the algorithm, as they have an influence on the evolution of the sizes of the pairs (q_{i+1}, q_i) which may grow due these extra factors. As these extra factors may only be powers of 2, they are easily expressed with the dyadic absolute value on $\mathbb{Q}$, at least when the input is rational. However, when extending the algorithm into a dynamical system on the unit interval, we lose track of these factors.

The (natural) idea is thus to add to the usual *CL* dynamical system (on the unit interval $\mathcal{I}$) a new component in the dyadic field $\mathbb{Q}_2$. The dyadic component is just added here to deal with the extra dyadic factors, as a sort of accumulator, but it is the former real component that dictates the evolution of the system. As the initial *CL* system has nice properties, the mixed system inherits this good behaviour. In particular, the transfer operator of the mixed system presents a dominant eigenvalue, and the dynamical analysis may be performed successfully. The constant H in Eq. (2) is actually the entropy of this extended system.

After this extension, the analysis follows classical steps, with methodology mixing tools from analytic combinatorics (generating functions, here of Dirichlet type), Tauberian theorems (relating the singularities of these generating functions to the asymptotics of their coefficients), functional analysis (which transfers the geometry of the dynamical system into spectral properties of the transfer operator).

Plan of the paper and notation. The paper is structured into three sections. Section 2 introduces the algorithm and its associated dynamical system, as well as the probabilistic model, the costs of interest and their generating functions. Then, Sect. 3 defines the extended dynamical system, allowing us to work with dyadic costs; it explains how the corresponding transfer operator provides alternative expressions for the generating functions. Finally, Sect. 4 describes the properties of the transfer operator, namely its dominant spectral properties on a convenient functional space. With Tauberian theorem, it provides the final asymptotic estimates for the mean values of the main costs of interest.

For an integer q, $\delta(q)$ denotes the dyadic valuation, i.e., is the greatest integer k for which 2^k divides q. The dyadic norm $|\cdot|_2$ is defined on $\mathbb{Q}$ with the equality $|a/b|_2 := 2^{\delta(b)-\delta(a)}$. The dyadic field $\mathbb{Q}_2$ is the completion of $\mathbb{Q}$ for this norm. See [7] for more details about the dyadic field $\mathbb{Q}_2$.

2 The *CL* Algorithm and Its Dynamical System

We kick off this section with a precise description of the *CL* algorithm, followed by its extension to the whole of the unit interval $\mathcal{I}$, giving rise to a dynamical system, called the *CL* system, whose inverse branches capture all of our costs of interest. Then we recall the already known features of the *CL* system and we present the probabilistic model, with its generating functions.

Description of the algorithm. The algorithm, described by Shallit in [8], is a sequence of (pseudo)–divisions: each division associates to a pair (p, q)[1] with $p < q$ a new pair (r, p') (where r is the "remainder") defined as follows

$$q = 2^a p + r, \quad p' = 2^a p, \quad \text{with} \quad a = a(p,q) := \max\{k \geq 0 \mid 2^k p \leq q\}.$$

[1] Our notations are not the same as in the paper of Shallit as we reverse the roles of p and q.

This transformation rewrites the old pair (p, q) in terms of the new one $(r, 2^a p)$ in matrix form,

$$\begin{pmatrix} p \\ q \end{pmatrix} = N_a \begin{pmatrix} r \\ 2^a p \end{pmatrix}, \quad \text{with } N_a = \begin{pmatrix} 0 & 2^{-a} \\ 1 & 1 \end{pmatrix} = 2^{-a}\, M_a, \quad M_a = \begin{pmatrix} 0 & 1 \\ 2^a & 2^a \end{pmatrix}. \tag{3}$$

The *CL* algorithm begins with the input (p, q) with $p < q$. It lets $(q_1, q_0) := (p, q)$, then performs a sequence of divisions

$$\left(q_{i+1}, q_i\right)^T = N_{a_{i+1}}\ \left(q_{i+2}, 2^{a_{i+1}}\ q_{i+1}\right)^T,$$

and stops after $k = K(p, q)$ steps on a pair of the form $(0, 2^{a_k} q_k)$. The complete execution of the algorithm uses the set of matrices N_a defined in (3), and writes the input as

$$\left(p, q\right)^T = N_{a_1} \cdot N_{a_2} \cdots N_{a_k}\ \left(0, 2^{a_k} q_k\right)^T.$$

The rational input p/q is then written as a continued fraction according to the LFTs (linear fractional transformations) h_a associated with matrices N_a or M_a,

$$\frac{p}{q} = \cfrac{2^{-a_1}}{1 + \cfrac{2^{-a_2}}{1 + \cfrac{2^{-a_3}}{1 + \ddots \cfrac{2^{-a_k}}{1}}}} = h_{a_1} \circ h_{a_2} \circ \cdots \circ h_{a_k}(0), \quad \text{with } h_a : x \mapsto \frac{2^{-a}}{1+x}. \tag{4}$$

Moreover, it is possible to choose the last exponent a_k to be 0. (and the last quotient to be 1). This is a gcd algorithm: as q_k is equal to $\gcd(p, q)$ up to a power of 2, the *CL* algorithm determines the odd part of $\gcd(p, q)$ whereas the even part is directly determined by the dyadic valuations of p and q.

Shallit [8] proves that this algorithm indeed terminates and characterizes the worst-case complexity of the algorithm. Figure 1 describes the execution of the algorithm on the pair $(31, 75)$.

Dynamical system. The relations

$$(r, 2^a p)^T = N_a^{-1}(p, q)^T, \quad (p, q)^T = N_a(r, 2^a p)^T,$$

are first transformed into relations on the associated rationals $p/q, r/(2^a p)$ via the LFT's T_a, h_a associated to matrices N_a^{-1}, N_a,

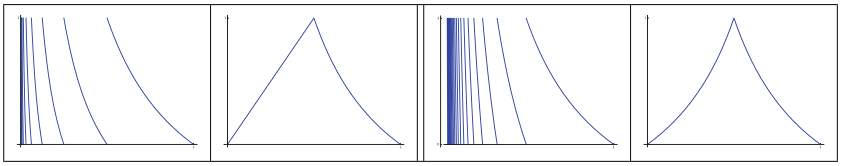
$$T_a(x) := \frac{1}{2^a x} - 1, \quad h_a(x) = \frac{1}{2^a(1+x)}, \quad a \geq 0. \tag{5}$$

They are then extended to the reals of the unit interval $\mathcal{I} := [0, 1]$. This gives rise to a dynamical system $(\mathcal{I}, T)$, denoted *CL* in the sequel, defined on the unit interval $\mathcal{I}$, with fundamental intervals $\mathcal{I}_a := [2^{-a-1}, 2^{-a}]$, the surjective branches $T_a \colon \mathcal{I}_a \to \mathcal{I}$, and their inverses $h_a \colon \mathcal{I} \to \mathcal{I}_a$.

i	a_i	$2^{a_i}q_i$	q_{i+1}	$(2^{a_i}q_i)_2$	$(q_{i+1})_2$	$\delta(2^{a_i}q_i)$	$\delta(q_{i+1})$	$\delta(\widehat{g}_i)$
0	–	75	31	1001011	11111	0	0	0
1	1	62	13	0111110	1101	1	0	0
2	2	52	10	110100	1010	2	1	1
3	2	40	12	101000	1100	3	2	2
4	1	24	16	11000	10000	3	4	3
5	0	16	8	10000	1000	4	3	3
6	0	8	8	1000	1000	3	3	3
7	–	8	0	1000	0	3	∞	3

Fig. 1. Execution for the input $(31, 75)$. Here $\widehat{g}_i = \gcd(2^{a_i}q_i, q_{i+1})$. The dyadic valuation $\delta(\widehat{g}_i)$ seems to linearly increase with i, with $\delta(\widehat{g}_i) \sim \delta(q_{i+1}) \sim i/2$ $(i \to \infty)$.

The *CL* system $(\mathcal{I}, T)$ is displayed on the left of the figure below, along with the shift $S : \mathcal{I} \to \mathcal{I}$ which gives rise to the *CL* system by induction on the first branch. The map S is a mix of the Binary and Farey maps, as its first branch comes from the Binary system, and the second one from the Farey system. On the right, the usual Euclid dynamical system (defined from the Gauss map) is derived from the Farey shift by induction on the first branch.



With each k-uple $\boldsymbol{a} := \langle a_1, a_2, \ldots, a_k\rangle \in \mathbb{N}^k$ we associate the matrix $M_{\boldsymbol{a}} := M_{a_1} \cdots M_{a_k}$ and the LFT $h_{\boldsymbol{a}} := h_{a_1} \circ h_{a_2} \circ \cdots \circ h_{a_k}$. Then the set $\mathcal{H}$ of the inverse branches, and the set $\mathcal{H}^k$ of the inverse branches of T^k (of depth k) are

$$\mathcal{H} := \{h_a \mid a \geq 0\}, \quad \mathcal{H}^k := \{h_{\boldsymbol{a}} \mid \boldsymbol{a} \in \mathbb{N}^k\}.$$

As the branches T_a are surjective, the inverse branches are defined on $\mathcal{I}$ and the images $h_a(\mathcal{I})$ for $a \geq 0$ form a topological partition of $\mathcal{I}$. This will be true at any depth, and the intervals $h_{\boldsymbol{a}}(\mathcal{I})$, called fundamental intervals of depth k, form a topological partition for $\boldsymbol{a} \in \mathbb{N}^k$.

Properties of the* CL *system. The Perron Frobenius operator

$$\mathbf{H}[f](x) := \sum_{h\in\mathcal{H}} |h'(x)|\, f(h(x)) = \left(\frac{1}{1+x}\right)^2 \sum_{a\geq 0} 2^{-a} f\left(\frac{2^{-a}}{1+x}\right). \tag{6}$$

describes the evolution of densities: If f is the initial density, $\mathbf{H}[f]$ is the density after one iteration of the system $(\mathcal{I}, T)$. The invariant density ψ is a fixed point for $\mathbf{H}$ and satisfies the functional equation

$$\psi(x) = \left(\frac{1}{1+x}\right)^2 \sum_{a\geq 0} 2^{-a}\, \psi\left(\frac{2^{-a}}{1+x}\right). \tag{7}$$

Chan [3] obtains an explicit form for ψ

$$\psi(x) = \frac{1}{\log(4/3)} \frac{1}{(x+1)(x+2)}. \tag{8}$$

He also proves that the system is ergodic with respect to ψ, and entropic. However, he does not provide an explicit expression for the entropy. We obtain here such an expression, with a precise study of the transfer operator of the system.

We introduce two (complex) parameters t, v in (6), and deal with a perturbation of the operator $\mathbf{H}$, defined by

$$\mathbf{H}_{t,v}[f](x) := \sum_{h \in \mathcal{H}} |h'(x)|^t \, d(h)^v \, f(h(x)) = \left(\frac{1}{1+x}\right)^{2t} \sum_{a \geq 0} 2^{\,a(v-t)} f\left(\frac{2^{-a}}{1+x}\right). \tag{9}$$

Such an operator $\mathbf{H}_{t,v}$ is called a transfer operator. When (t,v) satisfies $\Re(t-v) > 0$, we prove the following: the operator $\mathbf{H}_{t,v}$ acts nicely on the space $\mathcal{C}^1(\mathcal{I})$ endowed with the norm $|\cdot|_{1,1}$, defined by $|f|_{1,1} := |f|_0 + |f'|_0$, where $|\cdot|_0$ denotes the sup norm. In particular, it has a dominant eigenvalue $\lambda(t,v)$ separated from the remainder of the spectrum by a spectral gap, for (t,v) close to $(1,0)$. The Taylor expansion of $\lambda(t,v)$ near $(1,0)$

$$\lambda(t,v) \approx 1 - A(t-1) + Dv$$

involves the two constants $A = -\partial\lambda/\partial t(1,1,0), D = \partial\lambda/\partial v(1,1,0)$, that are expressed as mean values with respect to the invariant density ψ,

$$A = E - D, \quad E = \mathbb{E}_\psi[2|\log x|], \quad D = (\log 2)\, \mathbb{E}_\psi[\underline{a}], \tag{10}$$

(here, the function $\underline{a}$ associates with x the integer defined with the Iverson bracket $\underline{a}(x) := a \cdot [\![x \in h_a(\mathcal{I})]\!]$. The constants A is the entropy of the system, and E, D admit explicit expressions

$$E = \frac{1}{\log(4/3)} \left[\frac{\pi^2}{6} + 2 \sum_{k \geq 1} \frac{(-1)^k}{k^2\, 2^k} \right], \quad D = (\log 2) \frac{\log(3/2)}{\log(4/3)}. \tag{11}$$

Then, with (10) and (11), there is an explicit value for the entropy A, and

$$A \doteq 1.62352\ldots, \quad D \doteq 0.97693\ldots, \quad E \doteq 2.60045\ldots.$$

Main costs associated to a truncated expansion. Each real number of the unit interval admits an infinite continued fraction expansion derived from the dynamical system, which we call its *CLCF* expansion. When truncated at a finite depth, its expansion becomes finite, as in (4), and defines a LFT $h := h_{\boldsymbol{a}}$. This expansion gives rise to a rational p/q, (assumed to be irreducible) which is thus written as $p/q = h_{\boldsymbol{a}}(0)$.

On the other hand, the k-uple $\boldsymbol{a}$ defines a matrix $M_{\boldsymbol{a}}$ and an integer pair (P, Q), called the continuant pair, defined by $(P, Q)^T := M_{\boldsymbol{a}}(0, 1)^T$. The equality $P/Q = p/q$, holds, but, as the integers P and Q are not necessarily coprime, the pair (P, Q) does not coincide with the pair (p, q). The integer $R(Q) := Q/\gcd(P, Q)$, called the reduced continuant, is an important parameter that actually dictates the quality of the rational approximation given by the truncation of the *CLCF*. We will see that it also plays a central role in the analysis of the *CL* algorithm. As $\gcd(P, Q)$ divides $|\det(M_{\boldsymbol{a}})|$ that is a power of two, it is itself a power of two. It then proves fundamental to deal with dyadic tools.

The main interesting costs associated with a finite expansion, as in (4), are defined via the LFT h and mainly involve the continuant pair (P, Q) together with the absolute value of the determinant of the LFT h, denoted as $d(h)$. The next result describes these costs and provides alternative expressions.

Proposition 1. *Consider the function $G_2 : \mathbb{Q}_2 \to \mathbb{R}^+$ (called the* gcd *map) equal to $G_2(y) = \min(1, |y|_2^{-2})$, namely*

$$G_2(y) = 1 \quad \textit{for } |y|_2 \le 1, \qquad G_2(y) = |y|_2^{-2} \quad \textit{for } |y|_2 > 1. \tag{12}$$

The main costs associated with the CLCF *expansion of a rational $h(0)$*

$$Q, \quad g(P, Q) := \gcd(P, Q), \quad R(P, Q) = Q/\gcd(P, Q), \quad |Q|_2.$$

are all expressed in terms of the quadruple $(|h'(0)|, |h'(0)|_2, d(h), G_2[h(0)]$ *as*

$$Q^{-2} = |h'(0)|/d(h), \quad |Q|_2^{-2} = d(h)\,|h'(0)|_2,$$

$$R^{-2}(Q) = |h'(0)|\,|h'(0)|_2\,G_2[h(0)], \quad g^2(P, Q) = d(h)\,|h'(0)|_2\,G_2[h(0)].$$

Proof. One has (by definition)

$$P/Q = h(0), \quad Q^{-2} = |h'(0)|/d(h), \quad r(Q)^{-2} = g^2(P, Q)/Q^2.$$

As $g(P, Q)$ is a power of 2, and using the function G_2 defined in (12), one has

$$g^2(P, Q) = \min(|P|_2, |Q|_2)^{-2} = |Q|_2^{-2} \min(1, |P/Q|_2^{-2}) = |Q|_2^{-2} G_2(P/Q).$$

We conclude with the equalities: $|Q|_2^{-2} = |h'(0)|_2/|d(h)|_2, \; d(h) \cdot |d(h)|_2 = 1$.

Any cost C of Proposition 1 admits an expression of the form

$$|h'(0)|^t \; |h'(0)|_2^u \; d(h)^v \; G_2[h(0]^z.$$

The quadruple (t, u, v, z) associated with the cost C is denoted as γ_C. Moreover, as these costs C are expected to be of exponential growth with respect to the depth of the *CF*, we will work with their logarithms $c = \log C$. Figure 2 summarizes the result.

Cost C	$c = \log C$	Quadruple γ_C	Constant $M(c)$	Numerical value of $M(c)$
$d(h)$	σ	$(0, 0, 1, 0)$	D	$\doteq 0.97693\ldots$
Q^2	q	$(-1, 0, 1, 0)$	$A + D$	$\doteq 2.60045\ldots$
$g^2(P, Q)$	ϱ	$(0, 1, 1, 1)$	$B + D$	$\doteq 1.26071\ldots$
$R^2(P, Q)$	r	$(-1, -1, 0, -1)$	$A - B$	$\doteq 1.33973\ldots$
$\lvert Q\rvert_2^{-2}$	q_2	$(0, 1, 1, 0)$	$B + D$	$\doteq 1.26071\ldots$

Fig. 2. Main costs of interest, with their quadruple, and the constants which intervene in the analysis of their mean values. (see Theorem 1).

Generating functions. We deal with sets of coprime[2] integer pairs

$$\Omega := \{(p, q) \mid 0 < p < q,\ \gcd(p, q) = 1\}, \quad \Omega_N := \Omega \cap \{(p, q) \mid q \le N\}.$$

The set Ω_N is endowed with the uniform measure, and we wish to study the mean values $\mathbb{E}_N[c]$ of parameters c on Ω_N. We focus on parameters which describe the execution of the algorithm and are "read" from the $CF(p/q)$ built by the algorithm as in (4). They are defined in Proposition 1 and depend on the continuant pair (P, Q); as already explained, the reduced continuant $R(P, Q)$ plays a fundamental role here.

We deal with analytic combinatorics methodology and work with (Dirichlet) generating functions (dgf in short). Here is the plain Dirichlet generating function

$$S(s) := \sum_{(p,q)\in\Omega} \frac{1}{q^{2s}} = \frac{\zeta(2s-1)}{\zeta(2s)}. \tag{13}$$

There are also two generating functions that are associated with a cost $C : \Omega \to \mathbb{R}^+$ (and its logarithm c), namely the bivariate dgf and the cumulative dgf,

$$S_C(s, w) := \sum_{(p,q)\in\Omega} \frac{e^{wc(p,q)}}{q^{2s}}, \ \widehat{S}_C(s) := \sum_{(p,q)\in\Omega} \frac{c(p, q)}{q^{2s}} = \frac{\partial}{\partial w} S_C(s, w)\Big|_{w=0}. \tag{14}$$

The expectation $\mathbb{E}_N[c]$ is now expressed as a ratio which involves the sums $\Phi_N(S)$, $\Phi_N(\widehat{S}_C)$ of the first N coefficients of the Dirichlet series $S(s)$ and $\widehat{S}_C(s)$,

$$E_N[c] = \Phi_N[\widehat{S}_C]/\Phi_N[S]\,. \tag{15}$$

From principles of Analytic Combinatorics, we know that the dominant singularity of a dgf (here its singularity of largest real part) provides precise information (via notably its position and its nature) about the asymptotics of its coefficients, here closely related to the mean value $\mathbb{E}_N[c]$ via Eq. (15). Here, in the

[2] This restriction can be easily removed and our analysis extends to the set of all integer pairs.

Dirichlet framework, this transfer from the analytic behaviour of the dgf to the asymptotics of its coefficients is provided by Delange's Tauberian Theorem [5].

We now describe an alternative expression of these series, from which it is possible to obtain information regarding the dominant singularity, which will be transfered to the asymptotics of coefficients.

Proposition 2. *The Dirichlet generating $S(s)$ and its bivariate version $S_C(s,w)$ relative to a cost $C : \Omega \to \mathbb{R}$, admit alternative expressions*[3]

$$S(s) = S_C(s,0), \quad S_C(s,w) = \sum_{h \in \mathcal{H}^\star \cdot J} e^{wC(h)}\ |h'(0)|^s\ |h'(0)|_2^s\ G_2^s \circ h(0).$$

For any cost C described in Fig. 2, the general term of $S_C(s,w)$ is of the form

$$|h'(0)|^t\ |h'(0)|_2^u\ d(h)^v\ G_2^z \circ h(0),$$

and involves a quadruple of exponents (t,u,v,z), denoted as $\gamma_C(s,w)$, that is expressed with the quadruple γ_C defined in Fig. 2 as

$$\gamma_C(s,w) = s(1,1,0,1) + w\,\gamma_C. \tag{16}$$

Proof. By definition, the denominator q equals $R(P,Q)$. With Fig. 2, the quadruple relative to q^{-2s} is then $s(1,1,0,1)$, whereas the quadruple relative to $e^{wc} = C^w$ is just $w\gamma_C$.

We have thus described the general framework of our paper. We now look for an alternative form for the generating functions: in dynamical analysis, one expresses the dgf in terms of the transfer operator of the dynamical system which underlies the algorithm. Here, it is not possible to obtain such an alternative expressions if we stay in the real "world". This is why we will add a component to the *CL* system which allows us to express parameters with a dyadic flavour. It will be possible to express the dgf's in term of a (quasi-inverse) of an (extended) transfer operator, and relate their dominant singularity to the dominant eigenvalue of this extended transfer operator.

We then obtain our main result, precisely stated in Theorem 1, at the end of the paper: we will prove that the mean values $E_N[\log C]$ associated with our costs of interest are all of order $\Theta(\log N)$, and satisfy precise asymptotics that involve three constants A, B, D: the constants D and A come from the real word, and have been previously defined in (11) and (10), but there arises a new constant B that comes from the dyadic word.

3 The Extended Dynamical System

In this section, we extend the *CL* dynamical system, adding a new component to study the dyadic nature of our costs. We then introduce transfer operators, and express the generating functions in terms of the quasi-inverses of the transfer operators.

[3] We recall that the last exponent is 0 by convention, and the last LFT is thus $J = h_0$.

Extension of the dynamical system. We will work with a two-component dynamical system: its first component is the initial CL system, to which we add a second (new) component which is used to "follow" the evolution of dyadic phenomena during the execution of the first component.

We consider the set $\underline{\mathcal{I}} := \mathcal{I} \times \mathbb{Q}_2$. We define a new shift $\underline{T} : \underline{\mathcal{I}} \to \underline{\mathcal{I}}$ from the characteristics of the old shift T defined in (5). As each branch T_a, or its inverse h_a, is a LFT with rational coefficients, it is well-defined on $\mathbb{Q}_2$; it is moreover a bijection from $\mathbb{Q}_2 \cup \{\infty\}$ to $\mathbb{Q}_2 \cup \{\infty\}$. Then, each branch $\underline{T}_a$ of the new shift $\underline{T}$ is defined via the equality $\underline{T}_a(x, y) := (T_a(x), T_a(y))$ on the fundamental domain $\underline{\mathcal{I}}_a := \mathcal{I}_a \times \mathbb{Q}_2$, and the shift $\underline{T}_a$ is a bijection from $\underline{\mathcal{I}}_a$ to $\underline{\mathcal{I}} := \mathcal{I} \times \mathbb{Q}_2$ whose inverse branch $\underline{h}_a \colon (x, y) \mapsto (h_a(x), h_a(y))$ is a bijection from $\underline{\mathcal{I}}$ to $\underline{\mathcal{I}}_a$.

Measures. We consider the three domains

$$\mathcal{B} := \mathbb{Q}_2 \cap \{|y|_2 < 1\},\ \mathcal{U} := \mathbb{Q}_2 \cap \{|y|_2 = 1\},\ \mathcal{C} := \mathbb{Q}_2 \cap \{|y|_2 > 1\}.$$

There exists a Haar measure ν_0 on $\mathbb{Q}_2$ which is finite on each compact of $\mathbb{Q}_2$, and can be normalized with $\nu_0(\mathcal{B}) = \nu_0(\mathcal{U}) = 1/3$ (see [7]). We will deal with the measure ν with density G_2 wrt to ν_0, for which $\nu(\mathcal{C}) = 1/3$. The measure $\nu[2^k\mathcal{U}]$ equals $(1/3)2^{-|k|}$ for any $k \in \mathbb{Z}$ and ν is a probability measure on $\mathbb{Q}_2$. On $\underline{\mathcal{I}}$, we deal with the probability measure $\rho := \mu \times \nu$ where μ is the Lebesgue measure on $\mathcal{I}$ and ν is defined on $\mathbb{Q}_2$ as previously.

For integrals which involve a Haar measure, there is a change of variables formula[4]. As ν_0 is a Haar measure, and $d\nu = G_2\, d\nu_0$, this leads to the following change of variables formula, for any $F \in L^1(\mathbb{Q}_2, \nu)$,

$$\int_{\mathbb{Q}_2} |h'(y)|_2\ F(h(y)) \left[\frac{G_2(h(y))}{G_2(y)}\right]\ d\nu(y) = \int_{\mathbb{Q}_2} F(y)\ d\nu(y). \tag{17}$$

Density transformer and transfer operator. We now consider the operator $\underline{\mathbf{H}}$ defined as a "density transformer" as follows: with a function $F \in L^1(\underline{\mathcal{I}}, \rho)$, it associates a new function defined by

$$\underline{\mathbf{H}}\,[F](x, y) := \sum_{h \in \mathcal{H}} |h'(x)|\ |h'(y)|_2\ F(h(x), h(y))\ \left[\frac{G_2(h(y))}{G_2(y)}\right].$$

When F is a density in $L^1(\underline{\mathcal{I}}, \rho)$, then $\underline{\mathbf{H}}[F]$ is indeed the new density on $\underline{\mathcal{I}}$ after one iteration of the shift $\underline{T}$. This just follows from the change of variables formula (17) applied to each inverse branch $h \in \mathcal{H}$.

Proposition 2 leads us to a new operator that depends on a quadruple (t, u, v, z),

$$\mathbf{H}_{t,u,v,z}[F](x, y) := \sum_{h \in \mathcal{H}} |h'(x)|^t\ |h'(y)|_2^u\ d(h)^v F(h(x), h(y))\ \left[\frac{G_2(h(y))}{G_2(y)}\right]^z. \tag{18}$$

[4] This general result can be found for instance in Bourbaki [2], Chap. 10, p. 36.

We will focus on costs described in Fig. 2: we thus deal with operators associated with quadruples $\gamma_C(s,w)$ defined in Proposition 2, and in particular with the quadruple $(s,s,0,s)$, and its associated operator $\underline{\mathbf{H}}_s := \mathbf{H}_{s,s,0,s}$.

Alternative expressions of the Dirichlet generating functions. We start with the expressions of Proposition 1, consider the three types of dgf defined in (13) and (14), use the equality $G_2(0) = 1$, and consider the operator $\underline{\mathbf{J}}_s$ relative to the branch J used in the last step. For the plain dgf in (13), we obtain

$$S(s) = \sum_{h \in \mathcal{H}^{\star} \cdot J} |h'(0)|^s \ |h'(0)|_2^s \ G_2^s \circ h(0) = \underline{\mathbf{J}}_s \circ (I - \underline{\mathbf{H}}_s)^{-1}[1](0,0), \tag{19}$$

We now consider the bivariate dgf's defined in (14). For the depth K, one has

$$S_K(s,w) = e^w \underline{\mathbf{J}}_s \circ (I - e^w \underline{\mathbf{H}}_s)^{-1}[1](0,0);$$

For costs C of Fig. 2, the bivariate dgf involves the quasi-inverse of $\mathbf{H}_{\gamma_C(s,w)}$,

$$S_C(s,w) = \mathbf{J}_{\gamma_C(s,w)} \circ (I - \mathbf{H}_{\gamma_C(s,w)})^{-1}[1](0,0),$$

except for $C = |Q|_2^{-2}$, where the function 1 is replaced by the function G_2^w.

The dgf $\widehat{S}_C(s)$ defined in (14) is obtained with taking the derivative of the bivariate dgf wrt w (at $w = 0$); it is thus written with a double[5] quasi inverse which involves the plain operator $\underline{\mathbf{H}}_s$, separated "in the middle" by the cumulative operator $\underline{\mathbf{H}}_{s,(C)}$, namely

$$\widehat{S}_C(s) \asymp \underline{\mathbf{J}}_s \circ (I - \underline{\mathbf{H}}_s)^{-1} \circ \underline{\mathbf{H}}_{s,(C)} \circ (I - \underline{\mathbf{H}}_s)^{-1} \ [1](0,0), \tag{20}$$

and the cumulative operator is itself defined by $\underline{\mathbf{H}}_{s,(C)} := \frac{\partial}{\partial w} \mathbf{H}_{\gamma_C(s,w)} \Big|_{w=0}$.

4 Functional Analysis

This section deals with a delicate context, which mixes the specificities of each world –the real one, and the dyadic one–. It is devoted to the study of the quasi-inverses $(I - \underline{\mathbf{H}}_s)^{-1}$ intervening in the expressions of the generating functions of interest. We first define an appropriate functional space on which we prove the operators to act and admit dominant spectral properties. This entails that the quasi-inverse $(I - \underline{\mathbf{H}}_s)^{-1}$ admits a pole at $s = 1$, and we study its residue, which gives rise to the constants that appear in the expectations of our main costs.

Functional space. The delicate point of the dynamical analysis is the choice of a good functional space, that must be a subset of $L^1(\underline{\mathcal{I}}, \rho)$. Here, we know that, in the initial CL system, the transfer operator $\mathbf{H}_s$ acts in a good way on $\mathcal{C}^1(\mathcal{I})$. Then, for a function F defined on $\underline{\mathcal{I}}$, the main role will be played by the family

[5] There is another term which involves only a quasi-inverse. It does not intervene in the analysis.

of "sections" $\tilde{F}_y : x \mapsto \max(1, \log |y|_2)\, F(x,y)$ which will be asked to belong to $\mathcal{C}^1(\mathcal{I})$, under the norm $|\cdot|_{1,1}$, defined as $|\tilde{F}_y|_{1,1} := |\tilde{F}_y|_0 + |\tilde{F}_y|_1$ with

$$|\tilde{F}_y|_0 := \sup_{x\in\mathcal{I}} |\tilde{F}(x,y)|, \quad |\tilde{F}_y|_1 := \sup_{x\in\mathcal{I}} \left| \frac{\partial}{\partial x} \tilde{F}(x,y) \right| .$$

We work on the Banach space

$$\mathcal{F} := \left\{ F : \underline{\mathcal{I}} \to \mathbb{C} \mid F_y \in \mathcal{C}^1(\mathcal{I}), \quad y \mapsto \tilde{F}_y \text{ bounded} \right\},$$

endowed with the norm $||F|| := ||F||_0 + ||F||_1$, with

$$||F||_0 := \int_{\mathbb{Q}_2} |\tilde{F}_y|_0 \; d\nu(y), \quad ||F||_1 := \int_{\mathbb{Q}_2} |\tilde{F}_y|_1 \; d\nu(y). \tag{21}$$

The next Propositions 3, 4 and 5 will describe the behaviour of the operator $\mathbf{H}_{t,u,v,z}$ on the functional space $\mathcal{F}$. Their proofs are quite technical and are omitted here.

The first result exhibits a subset of quadruples (t,u,v,z) which contains $(1,1,0,1)$ for which the resulting operator $\mathbf{H}_{t,u,v,z}$ acts on $\mathcal{F}$.

Proposition 3. *The following holds:*

(a) When the complex triple (t,u,v) satisfies the constraint $\Re(t-v-|u-1|) > 0$, the operator $\mathbf{H}_{t,u,v,u}$ acts on $\mathcal{F}$ and is analytic with respect to the triple (t,u,v).
(b) The operator $\underline{\mathbf{H}}_s := \mathbf{H}_{s,s,0,s}$ acts on $\mathcal{F}$ for $\Re s > 1/2$, and the norm $||\cdot||_0$ of the operator $\underline{\mathbf{H}}_s$ satisfies $||\underline{\mathbf{H}}_s||_0 < 1$ for $\Re s > 1$.

Dominant spectral properties of the operator. The next result describes some of the main spectral properties of the operator on the space $\mathcal{F}$. Assertion (a) entails that the k-th iterate of the operator behaves as a true k-th power of its dominant eigenvalue. Then, as stated in (c), its quasi-inverse behaves as a true quasi-inverse which involves its dominant eigenvalue.

Proposition 4. *The following properties hold for the operator $\mathbf{H}_{t,u,v,u}$, when the triple (t,u,v) belongs to a neighborhood $\mathcal{V}$ of $(1,1,0)$.*

(a) There is a unique dominant eigenvalue, separated from the remainder of the spectrum by a spectral gap, and denoted as $\lambda(t,u,v)$, with a (normalized) dominant eigenfunction $\Psi_{t,u,v}$ and a dominant eigenmeasure $\rho_{t,u,v}$ for the dual operator.
(b) At $(t,u,v,u) = (1,1,0,1)$, the operator $\mathbf{H}_{t,u,v,u}$ coincides with the density transformer $\underline{\mathbf{H}}_1$. At $(1,1,0)$ the dominant eigenvalue $\lambda(t,u,v)$ equals 1, the function $\Psi_{t,u,v}$ is the invariant density Ψ and the measure $\rho_{t,u,v}$ equals the measure ρ.

(c) The estimate holds for any function $F \in L^1(\underline{\mathcal{I}}, \rho)$ with $\rho[F] \neq 0$,

$$(I - \mathbf{H}_{t,u,v,u})^{-1}[F](x,y) \sim \frac{\lambda(t,u,v)}{1-\lambda(t,u,v)}\, \Psi_{t,u,v}(x,y)\, \rho_{t,u,v}[F].$$

(d) For $\Re s = 1, s \neq 1$, the spectral radius of $\mathbf{H}_{s,s,0,s}$ is strictly less than 1.

The third result describes the Taylor expansion of $\lambda(t,u,v)$ at $(1,1,0)$, and makes precise the behaviour of the quasi-inverse described in (c).

Proposition 5. *The Taylor expansion of the eigenvalue $\lambda(t,u,v)$ at $(1,1,0)$, written as $\lambda(t,u,v) \sim 1 - A(t-1) + B(u-1) + Dv$, involves the constants*

$$A = -\partial\lambda/\partial t(1,1,0),\ B = \partial\lambda/\partial u(1,1,0),\ D = \partial\lambda/\partial v(1,1,0)$$

(a) The constants A and D already appear in the context of the plain dynamical system, and are precisely described in (11) and (10). In particular $A - D$ is equal to the integral $E := \mathbb{E}_\Psi[2|\log x|]$;
(b) The constant B is defined with the extension of the dynamical system and its invariant density $\Psi = \Psi_{1,1,0}$. The constant $B + D$ is equal to the dyadic analog E_2 of the integral E, namely, $B + D = E_2 := \mathbb{E}_\Psi[2\log|y|_2]$;
(c) The constant $A - B$ is the entropy of the extended dynamical system.

Final result for the analysis of the* CL *algorithm. We then obtain our final result:

Theorem 1. *The mean values $E_N[c]$ for $c \in \{K, \sigma, q, \varrho, r, q_2\}$ on the set Ω_N are all of order $\Theta(\log N)$ and admit the precise following estimates,*

$$E_N[K] \sim \frac{2}{H}\log N, \quad \mathbb{E}_N[c] \sim M(c)\cdot \mathbb{E}_N[K], \quad \textit{for } c \in \{\sigma, q, \varrho, r, q_2\}.$$

The constant H is the entropy of the extended system. The constants H and $M(c)$ are expressed with a scalar product that involves the gradient $\nabla\lambda$ of the dominant eigenvalue at $(1,1,0)$ and the beginning $\widehat{\gamma}_C$ of the quadruple γ_C associated with the cost c. More precisely

$$H = -\langle \nabla\lambda, (1,1,0)\rangle, \quad M(c) = \langle \nabla\lambda, \widehat{\gamma}_C\rangle.$$

The constants $M(c)$ are exhibited in Fig. 2.

Proof. Now, the Tauberian Theorem comes into play, relating the behaviour of a Dirichlet series $F(s)$ near its dominant singularity with asymptotics for the sum $\Phi_N(F)$ of its first N coefficients. Delange's Tauberian Theorem is stated as follows (see [5]):

Consider for $\sigma > 0$ a Dirichlet series $F(s) := \sum_{n\geq 1} a_n n^{-2s}$ with non negative coefficients which converges for $\Re s > \sigma$. Assume moreover:

(i) *$F(s)$ is analytic on $\{\Re s, s \neq \sigma\}$,*

(ii) *near* σ, $F(s)$ *satisfies* $F(s) \sim A(s)(s-\sigma)^{-(k+1)}$ *for some integer* $k \geq 0$.

Then, as $N \to \infty$, *the sum* $\Phi_N(F)$ *of its first* N *coefficients satisfies*

$$\Phi_N(F) := \sum_{n \leq N} a_n \sim 2^k\, A(\sigma)\, [\sigma\Gamma(k+1)]^{-1}\, N^{2\sigma}\, \log^k N.$$

We now show that the two dgf's $S(s)$ and $\widehat{S}_C(s)$ satisfy the hypotheses of the Tauberian theorem. The two expressions obtained in (19) and (20) involve quasi-inverses $(I - \underline{\mathbf{H}}_s)^{-1}$, a simple one in (19), a double one in (20).

First, Propositions 3(b) and 4(d) prove that such quasi-inverses are analytic on $\Re s \geq 1, s \neq 1$. Then Proposition 4(c), together with Eq. (20), shows that $S(s)$ and $\widehat{S}_C(s)$ have a pole at $s = 1$, of order 1 for $S(s)$, of order 2 for $\widehat{S}_C(s)$.

We now evaluate the dominant constants: first, the estimate holds,

$$1 - \lambda(s,s,0) \sim (A-B)(s-1) = H(s-1), \text{ with } H = -\langle \nabla\lambda, (1,1,0)\rangle.$$

Second, with Proposition 4(c), the dgf's $S(s)$ and $\widehat{S}_C(s)$ admit the following estimates which both involve the constant $a = \underline{\mathbf{J}}[\Psi](0,0)$, namely,

$$S(s) \sim \frac{a}{H(s-1)}, \qquad \widehat{S}_C(s) \sim \frac{a}{H^2(s-1)^2}\, \rho\left[\underline{\mathbf{H}}_{1,(C)}[\Psi]\right].$$

We now explain the occurrence of the constant $M(c)$: we use the definition of the triple $\widehat{\gamma}_C(s,w)$, the definition of the cumulative operator $\underline{\mathbf{H}}_{1,(C)}$ as the derivative of the bivariate operator $\mathbf{H}_{\gamma_C(s,w)}$ at $(s,w) = (1,0)$, and the fact that $\mathbf{H}_{1,1,0,1} = \underline{\mathbf{H}}_1$ is the density transformer. This entails the sequence of equalities,

$$\rho\left[\underline{\mathbf{H}}_{1,(C)}[\Psi]\right] = \frac{\partial}{\partial w}\lambda(\widehat{\gamma}_C(1,w))\Big|_{w=0} = \langle \nabla\lambda, \widehat{\gamma}_C\rangle = M(c).$$

About the constant B. The invariant density Ψ –more precisely the function $\widehat{\Psi} := \Psi \cdot G_2$ – satisfies a functional equation of the same type as the invariant function ψ, (described in Eq. (7)), namely,

$$\widehat{\Psi}(x,y) = \left(\frac{1}{1+x}\right)^2 \left|\frac{1}{1+y}\right|_2^2 \sum_{a\geq 0} \widehat{\Psi}\left(\frac{2^{-a}}{1+x}, \frac{2^{-a}}{1+y}\right).$$

Comparing to Eq. (7), we “lose” the factor 2^{-a} in the sum, and so we have not succeed in finding an explicit formula for Ψ. We do not know how to evaluate the integral E_2 defined in Proposition 5(b).

However, we conjecture[6] the equality $D - B = \log 2$, from experiments of the same type as those described in Fig. 1. This would entail an explicit value for the entropy of the extended system,

$$\frac{1}{2\log 2 - \log 3}\left[\frac{\pi^2}{6} + 2\sum_{k\geq 1}\frac{(-1)^k}{k^2\, 2^k} - (\log 2)(3\log 3 - 4\log 2)\right] \doteq 1.33973\ldots$$

[6] This will be explained in the long paper.

Conclusions and Extensions. We have studied the Continued Logarithm Algorithm and analyzed in particular the number of pseudo divisions, and the total number of shifts. It would be nice to obtain an explicit expression of the invariant density, that should entail a proven expression of the entropy of the dynamical system. It is also surely possible to analyze the bit complexity of the algorithm, notably in the case when one eliminates the rightmost zeroes when are shared by the two $q_i's$ (as suggested by Shallit). Such a version of this algorithm may have a competitive bit complexity that merits a further study.

There exist two other gcd algorithm that are based on binary shifts, all involving a dyadic point of view: the Binary Algorithm, and "the Tortoise and the Hare" algorithm, already analyzed in [4,9]; however, the role of the binary shifts is different in each case. The strategy of the present algorithm is led by the most significant bits, whereas the strategy of the "Tortoise and the Hare" is led by the least significant bits. The Binary algorithm adopts a mixed strategy, as it performs both right-shifts and subtractions. We have the project to unify the analysis of these three algorithms, and better understand the role of the dyadic component in each case.

References

1. Borwein, J.M., Hare, K.G., Lynch, J.G.: Generalized continued logarithms and related continued fractions. J. Integer Seq. **20**, 51 p. (2017). Article no. 17.5.7
2. Bourbaki, N.: Variétés différentielles et analytiques. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-34397-4
3. Chan, H.-C.: The asymptotic growth rate of random Fibonacci type sequences. Fibonacci Q. **43**(3), 243–255 (2005)
4. Daireaux, B., Maume-Deschamps, V., Vallée, B.: The Lyapounov tortoise and the dyadic hare. In: Proceedings of AofA 2005, DMTCS, pp. 71–94 (2005)
5. Delange, H.: Généralisation du Théorème d'Ikehara. Ann. Sc. ENS **71**, 213–242 (1954)
6. Gosper, B.: Continued fraction arithmetic (1978, unpublished manuscript)
7. Koblitz, N.: p-adic Numbers, p-adic Analysis and Zeta Functions, 2nd edn. Springer, New York (1984). https://doi.org/10.1007/978-1-4612-1112-9
8. Shallit, J.: Length of the continued logarithm algorithm on rational inputs (2016). https://arxiv.org/abs/1606.03881v2, arXiv:1606.03881v2
9. Vallée, B.: Dynamics of the binary Euclidean algorithm: functional analysis and operators. Algorithmica **22**(4), 660–685 (1998)
10. Vallée, B.: Euclidean dynamics. Discret. Continuous Dyn. Syst. **15**(1), 281–352 (2006)

Quadratic Simulations of Merlin–Arthur Games

Thomas Watson(✉)

Department of Computer Science, University of Memphis, Memphis, USA
Thomas.Watson@memphis.edu

Abstract. The known proofs of $\mathsf{MA} \subseteq \mathsf{PP}$ incur a quadratic overhead in the running time. We prove that this quadratic overhead is necessary for black-box simulations; in particular, we obtain an oracle relative to which $\mathsf{MA\text{-}TIME}(t) \not\subseteq \mathsf{P\text{-}TIME}(o(t^2))$. We also show that 2-sided-error Merlin–Arthur games can be simulated by 1-sided-error Arthur–Merlin games with quadratic overhead. We also present a simple, query complexity based proof (provided by Mika Göös) that there is an oracle relative to which $\mathsf{MA} \not\subseteq \mathsf{NP}^{\mathsf{BPP}}$ (which was previously known to hold by a proof using generics).

1 Introduction

There are several complexity class inclusions for which all the known proofs consist of "black-box" simulations incurring at least a quadratic overhead in the running time. There have also been lower bounds showing that for some of these inclusions, the quadratic overhead is necessary for black-box simulations (which also yields corresponding oracle separations). We begin by giving an overview of this topic. For convenience we abbreviate "quadratic-overhead black-box simulation" as "quadratic simulation". (Some relevant complexity class definitions can be found in the full version.)

- $\mathsf{BPP} \subseteq \Sigma_2\mathsf{P}$ [Sip83, Lau83] holds by quadratic simulations, and Viola [Vio09] proved that the quadratic overhead is necessary. Some known strengthenings of this inclusion include $\mathsf{S}_2{\cdot}\mathsf{BPP} \subseteq \mathsf{S}_2\mathsf{P}$ [RS98] and the facts that 2-sided-error Merlin–Arthur and Arthur–Merlin games are equivalent to their 1-sided-error counterparts: $\mathsf{MA}_2 \subseteq \mathsf{MA}_1$ and $\mathsf{AM}_2 \subseteq \mathsf{AM}_1$. Of course, the lower bound of [Vio09] also applies to these strengthenings.
- Arthur–Merlin games can simulate Merlin–Arthur games ($\mathsf{MA}_1 \subseteq \mathsf{AM}_1$ and $\mathsf{MA}_2 \subseteq \mathsf{AM}_2$) quadratically [Bab85]. Diehl [Die07] proved that the quadratic overhead is necessary, even for $\mathsf{MA}_1 \subseteq \mathsf{AM}_2$. (As a side result, we complement this by giving a quadratic simulation even for $\mathsf{MA}_2 \subseteq \mathsf{AM}_1$.)
- $\mathsf{MA}_2 \subseteq \mathsf{PP}$ [Ver92] holds by quadratic simulations. As our main result, we prove that the quadratic overhead is necessary (which was stated as an open

Watson. T—Supported by NSF grant CCF-1657377.

problem in [Die07]), even for the weaker inclusion $\mathsf{N{\cdot}coRP} \subseteq \mathsf{PP}$.[1] A strengthening of the latter inclusion is the quadratic simulation for $\mathsf{P{\cdot}BQP} \subseteq \mathsf{PP}$ [FR99].

- PP is closed under intersection by quadratic simulations [BRS95] (for all $L_1, L_2 \in \mathsf{P\text{-}TIME}(n)$ we have $L_1 \cap L_2 \in \mathsf{P\text{-}TIME}(n^2)$). Sherstov [She13] proved that the quadratic overhead is necessary.

1.1 Statement of Result

Consider the partial function $F_{\mathsf{N{\cdot}coR}}$ that takes a $2^n \times 2^n$ boolean matrix with the promise that each row has either all 1's or at most half 1's, and evaluates to 1 if there exists an all-1 row, and to 0 otherwise.

Theorem 1. *Every randomized unbounded-error decision tree for* $F_{\mathsf{N{\cdot}coR}}$ *uses either* $\Omega(n)$ *queries or* $2^{\Omega(n)}$ *random bits.*

For our interpretation about the necessity of a quadratic overhead (see the corollaries below), it suffices to have $\Omega(n^2)$ random bits (rather than $2^{\Omega(n)}$) at the end of the theorem statement.

Corollary 1. *There is an oracle relative to which* $\mathsf{N{\cdot}coR\text{-}TIME}(n) \not\subseteq \mathsf{P\text{-}TIME}(o(n^2))$.

Corollary 1 holds in the standard model of relativization where the oracle tape is erased after each query. This forces each query to cost linear time, which makes sense in our context since a query is intended to correspond to running a simulation of the deterministic algorithm underlying an $\mathsf{N{\cdot}coR\text{-}TIME}(n)$ algorithm. Corollary 1 follows in a completely routine way from Theorem 1 (see [Vio09, Die07] for examples of how such diagonalization arguments go).

Our result can also be interpreted in terms of what we call "black-box proofs of $\mathsf{N{\cdot}coR\text{-}TIME}(n) \subseteq \mathsf{P\text{-}TIME}(t)$". Such a proof consists of a uniform randomized algorithm that takes 1^n as input, computes $F_{\mathsf{N{\cdot}coR}}$ with unbounded error on an instance it has oracle access to, and runs in time $O(t(n))$ where each oracle query is charged time n. All known proofs of that inclusion are indeed black-box.

Corollary 2. *There is no black-box proof of* $\mathsf{N{\cdot}coR\text{-}TIME}(n) \subseteq \mathsf{P\text{-}TIME}(o(n^2))$.

Corollary 2 follows immediately from Corollary 1 since black-box proofs relativize. Corollary 2 also follows directly from Theorem 1 since such a black-box proof is just a uniform, time-efficient implementation of a randomized unbounded-error decision tree for $F_{\mathsf{N{\cdot}coR}}$ that uses $o(n)$ queries and $o(n^2)$ random bits.

[1] We mention that in the world of communication complexity, a nearly quadratic separation between $\mathsf{N{\cdot}coRP}$-type complexity and PP-type complexity is witnessed by the inner product mod 2 function—see [AW09] for the $\mathsf{N{\cdot}coRP}$ upper bound and [KN97, Sects. 3.5–3.6 and references therein] for the PP lower bound. However, this is not directly relevant to our results since the upper bound is really specific to communication complexity.

For convenience, we have focused on time n vs. n^2, but our lower bound also works for any time-constructible $t(n)$ vs. $t(n)^2$.

1.2 Relevance to Time-Space Lower Bounds

There is a line of research on time-space lower bounds for problems related to satisfiability [vM06]. It is known that for every constant $\epsilon > 0$,

(i) SAT (which is NP-complete) cannot be solved by a deterministic algorithm running in time $n^{2\cos(\pi/7)-\epsilon} \approx n^{1.8019}$ and space $n^{o(1)}$ [Wil08];
(ii) Σ_2SAT (which is $\Sigma_2\mathsf{P}$-complete) cannot be solved by a bounded-error randomized algorithm running in time $n^{2-\epsilon}$ and space $n^{o(1)}$ [Dv06];
(iii) MajMajSAT (which is $\mathsf{P{\cdot}PP}$-complete) cannot be solved by a bounded-error quantum algorithm running in time $n^{1+o(1)}$ and space $n^{1-\epsilon}$ [vW12, AKR+01].

It is open to prove a nontrivial randomized time-space lower bound for SAT rather than Σ_2SAT (the first rather than the second level of the polynomial hierarchy). A natural approach to prove this (following [Dv06]) would involve "swapping Arthur and Merlin" (i.e., using $\mathsf{MA}_2 \subseteq \mathsf{AM}_2$); however, the quadratic overhead is too inefficient to yield any nontrivial lower bound. Indeed, one of the motivations for the result of [Die07] is that it implies this approach cannot be made to work via a subquadratic black-box simulation.

Similarly, it is open to prove a nontrivial quantum time-space lower bound for MajSAT rather than MajMajSAT (the first rather than the second level of the counting hierarchy). A natural approach to prove this (following [vW12]) would involve "absorbing quantumness into a majority quantifier" (i.e., using $\mathsf{P{\cdot}BQP} \subseteq \mathsf{PP}$ [FR99]); however, the quadratic overhead is too inefficient to yield any nontrivial lower bound. Our result implies this approach cannot be made to work via a subquadratic black-box simulation (since $\mathsf{N{\cdot}coR\text{-}TIME}(n) \subseteq \mathsf{P{\cdot}BQ\text{-}TIME}(n)$).

2 Proof of Theorem 1

Suppose for contradiction that $F_{\mathsf{N\cdot coR}}$ has a randomized unbounded-error decision tree using $\leq n/6$ queries and $\leq 2^{n/4}$ uniformly random bits. Such a decision tree can be expressed as a polynomial threshold function (PTF) with integer coefficients, having degree $\leq n/6$ and weight $\leq 2^{2^{n/3}}$ (the weight is the sum of the absolute values of the coefficients). We use a two-step argument: first, we show that a particular approach for designing such a PTF fails; second, we essentially show that if *that* approach fails then *every* approach fails (by using an adaptation of Vereshchagin's machinery from [Ver95]).

If there were a univariate polynomial p of degree $\leq n/6$ such that $p(2^n) > 2^n$ and $p(i) \in [0,1]$ for all $i \in \{0,1,2,\ldots,2^{n-1}\}$, then we could get a PTF of degree $\leq n/6$ for $F_{\mathsf{N\cdot coR}}$ by taking the sum over all rows of p applied to the sum of the bits in that row, and using 2^n as the threshold. (Moreover, if the coefficients of p were all integer multiples of some $a > 0$ and p had weight $\leq a2^{2^{n/4}}$, then we could use p/a to get a PTF having weight $\leq 2^{2^{n/3}}$.) However, this approach cannot work:

Lemma 1. *There is no univariate polynomial p of degree $\leq n/6$ such that $p(2^n) > 2^{n/2}$ and $p(i) \in [0,1]$ for all $i \in \{0,1,2,\ldots,2^{n-1}\}$.*

Proof. Let us modify p by transforming the input interval $[0, 2^{n-1}]$ to $[-1,1]$ and shifting the graph down by $1/2$, i.e., define the polynomial $q(x) := p\big((x+1)2^{n-2}\big) - 1/2$. Then we have $q(3) > 2^{n/2} - 1/2$ and $q(-1 + i/2^{n-2}) \in [-1/2, 1/2]$ for all $i \in \{0,1,2,\ldots,2^{n-1}\}$. The latter property implies, by a standard result that has been widely used in the literature and is generally attributed to [EZ64, RC66], that for all $x \in [-1,1]$ we have $|q(x)| \leq (1/2)/\big(1 - O(\deg(q)^2/2^n)\big)$, which is at most 1 since $\deg(q) = \deg(p) \leq n/6 \leq o(2^{n/2})$.

In summary, $q(3) > 2^{n/2} - 1/2$, $|q(x)| \leq 1$ for all $x \in [-1,1]$, and $\deg(q) \leq n/6$. To show that this is impossible, we appeal to a classic result stating that Chebyshev polynomials are extremal in the following sense (see [Riv81, Theorem 1.10] or [Car, Theorem 4.12] for a proof): If T_d is the degree-d Chebyshev polynomial of the first kind (defined by the recurrence $T_0(x) := 1$, $T_1(x) := x$, and $T_{d+1}(x) := 2xT_d(x) - T_{d-1}(x)$ for $d \geq 1$) and q is any degree-d polynomial such that $|q(x)| \leq 1$ for all $x \in [-1,1]$, then for all $x \geq 1$ we have $|q(x)| \leq T_d(x)$. To get a contradiction, note that the recurrence trivially implies that $T_d(3) \leq 6^d$, and thus $q(3) \leq 6^d \leq 2^{n/2} - 1/2$ for $d \leq n/6$. □

Now we begin the bootstrapping.

Lemma 2. *There exist distributions D_0 and D_1 over $\{0,1,2,\ldots,2^{n-1}\} \cup \{2^n\}$ such that $\mathbb{P}_{D_0}[2^n] = 0$, $\mathbb{P}_{D_1}[2^n] = 2^{-n/2}$, and $\mathbb{E}_{i\sim D_0}[i^k] = \mathbb{E}_{i\sim D_1}[i^k]$ for all $k \in \{0,1,2,\ldots,n/6\}$.*

Proof. The lemma is equivalent to the feasibility of the following system with variables v_i and w_i for $i \in \{0,1,2,\ldots,2^{n-1}\}$ (representing $\mathbb{P}_{D_0}[i]$ and $\mathbb{P}_{D_1}[i]$ respectively), where we define $\delta := 2^{-n/2}$.

$$\begin{array}{rcll}
\sum_i v_i & = & 1 & \\
\sum_i w_i & = & 1-\delta & \\
\sum_i v_i \cdot i^k - \sum_i w_i \cdot i^k & = & \delta \cdot (2^n)^k & \quad\text{for all } k \in \{0,1,2,\ldots,n/6\} \\
v_i,\ w_i & \geq & 0 & \quad\text{for all } i \in \{0,1,2,\ldots,2^{n-1}\}
\end{array}$$

By Farkas's Lemma, this is equivalent to the infeasibility of the following system with variables x, y, and z_k for $k \in \{0,1,2,\ldots,n/6\}$.

$$\begin{array}{rcll}
x + \sum_k z_k \cdot i^k & \geq & 0 & \quad\text{for all } i \in \{0,1,2,\ldots,2^{n-1}\} \\
y - \sum_k z_k \cdot i^k & \geq & 0 & \quad\text{for all } i \in \{0,1,2,\ldots,2^{n-1}\} \\
x + y\cdot(1-\delta) + \sum_k z_k \cdot \delta \cdot (2^n)^k & < & 0 &
\end{array}$$

Defining the polynomial $Z(i) := \sum_k z_k \cdot i^k$, this system can be rewritten as follows.

$$Z(i) \in [-x, y] \qquad \text{for all } i \in \{0, 1, 2, \ldots, 2^{n-1}\} \tag{1}$$
$$x + y \cdot (1 - \delta) + \delta \cdot Z(2^n) < 0 \tag{2}$$

Suppose for contradiction this system is feasible; in particular $y \geq -x$. We cannot have $y = -x$ since then by (1), Z would either be the constant $y = -x$, thus violating (2), or have degree $>2^{n-1} > n/6$. Thus we may assume $x + y > 0$. If we define the polynomial $Z^*(i) := (y - Z(i))/(x + y)$ then $Z^*(i) \in [0, 1]$ for all $i \in \{0, 1, 2, \ldots, 2^{n-1}\}$ by (1), and $Z^*(2^n) > 1/\delta = 2^{n/2}$ by (2); yet $\deg(Z^*) = \deg(Z) \leq n/6$, contradicting Lemma 1. ◻

For $b \in \{0, 1\}$, define μ_b as the distribution over $2^n \times 2^n$ boolean matrices M obtained by, for each row independently, sampling $i \sim D_b$ and then taking a uniformly random length-2^n bit string of Hamming weight i. Let "$P(M) > t$" be the purported PTF for $F_{\mathsf{N\cdot coR}}$ (where t is an integer). The following two lemmas provide a contradiction.

Lemma 3. $\mathbb{E}_{\mu_1}[P(M)] > \mathbb{E}_{\mu_0}[P(M)]$.

Lemma 4. $\mathbb{E}_{\mu_1}[P(M)] = \mathbb{E}_{\mu_0}[P(M)]$.

Proof (of Lemma 3). Let us abbreviate $F_{\mathsf{N\cdot coR}}$ as F. Observe that $\mathbb{P}_{\mu_0}\big[F^{-1}(0)\big] = 1$ and $\mathbb{P}_{\mu_1}\big[F^{-1}(1)\big] = 1 - (1 - 2^{-n/2})^{2^n} \geq 1 - e^{-2^{n/2}}$. Also, notice that $|P(M)| \leq \text{weight}(P) \leq 2^{2^{n/3}}$ for all M; in particular, $t < 2^{2^{n/3}}$. Thus,

$$\begin{aligned}
&\mathbb{E}_{\mu_1}[P(M)]\\
&= \mathbb{E}_{\mu_1}\big[P(M) \,\big|\, F^{-1}(1)\big] \cdot \mathbb{P}_{\mu_1}\big[F^{-1}(1)\big] + \mathbb{E}_{\mu_1}\big[P(M) \,\big|\, F^{-1}(0)\big] \cdot \mathbb{P}_{\mu_1}\big[F^{-1}(0)\big]\\
&\geq (t + 1) \cdot \big(1 - e^{-2^{n/2}}\big) - 2^{2^{n/3}} \cdot e^{-2^{n/2}}\\
&> t\\
&\geq \mathbb{E}_{\mu_0}[P(M)].
\end{aligned}$$
◻

Proof (of Lemma 4). Define U_i to be the uniform distribution over length-2^n bit strings of Hamming weight i. For any $C \subseteq [2^n]$, we have $\mathbb{P}_{u \sim U_i}\big[u_C \text{ is all 1's}\big] = \frac{i(i-1)\cdots(i-|C|+1)}{2^n(2^n-1)\cdots(2^n-|C|+1)}$ (most easily seen by imagining that u is fixed and C is random); this is a polynomial of degree $|C|$ in i, which we write as $Q^{|C|}(i) := \sum_{k=0}^{|C|} Q_k^{|C|} \cdot i^k$. We also write $P(M) := \sum_S P_S \prod_{(r,c) \in S} M_{r,c}$ where the sum ranges over $S \subseteq [2^n] \times [2^n]$ with $|S| \leq n/6$. For a row index $r \in [2^n]$, let $S_r := \big\{c \in [2^n] : (r, c) \in S\big\}$. For each $b \in \{0, 1\}$ we have

$$\begin{aligned}
\mathbb{E}_{\mu_b}[P(M)] &= \textstyle\sum_S P_S\, \mathbb{P}_{\mu_b}\big[M_S \text{ is all 1's}\big]\\
&= \textstyle\sum_S P_S \prod_r \mathbb{E}_{i \sim D_b} \mathbb{P}_{u \sim U_i}\big[u_{S_r} \text{ is all 1's}\big]\\
&= \textstyle\sum_S P_S \prod_r \mathbb{E}_{i \sim D_b}\big[Q^{|S_r|}(i)\big]\\
&= \textstyle\sum_S P_S \prod_r \sum_k Q_k^{|S_r|} \mathbb{E}_{i \sim D_b}[i^k].
\end{aligned}$$

By Lemma 2, this value does not depend on b (since $k \leq |S_r| \leq |S| \leq n/6$ always holds). ◻

3 Quadratic Simulation for $\mathsf{MA}_2 \subseteq \mathsf{AM}_1$

Theorem 2. $\mathsf{MA}_2\text{-}\mathsf{TIME}(n) \subseteq \mathsf{AM}_1\text{-}\mathsf{TIME}(n^2)$.

$$\begin{tikzcd}
 & \mathsf{AM}_2 \arrow[dr] & \\
\mathsf{MA}_2 \arrow[ur] \arrow[rr] \arrow[dr] & & \mathsf{AM}_1 \\
 & \mathsf{MA}_1 \arrow[ur] \arrow[uu, dashed] &
\end{tikzcd}$$

The four diagonal arrows in the figure represent known simulations with quadratic overhead: $\mathsf{MA}_2\text{-}\mathsf{TIME}(n) \subseteq \mathsf{MA}_1\text{-}\mathsf{TIME}$ $(n^2 \operatorname{polylog} n)$, $\mathsf{AM}_2\text{-}\mathsf{TIME}(n) \subseteq \mathsf{AM}_1\text{-}\mathsf{TIME}$ $(n^2 \operatorname{polylog} n)$ follow by the "covering by shifts" argument of [Lau83], while $\mathsf{MA}_2\text{-}\mathsf{TIME}(n) \subseteq \mathsf{AM}_2\text{-}\mathsf{TIME}(n^2)$, $\mathsf{MA}_1\text{-}\mathsf{TIME}(n) \subseteq \mathsf{AM}_1\text{-}\mathsf{TIME}(n^2)$ follow by amplification and swapping the quantifiers, as shown in [Bab85]. The horizontal arrow represents Theorem 2. The dashed vertical arrow represents the lower bound of [Die07] showing that black-box or relativizing techniques cannot even yield $\mathsf{MA}_1\text{-}\mathsf{TIME}(n) \subseteq \mathsf{AM}_2\text{-}\mathsf{TIME}(o(n^2))$.

Of course, a 4th-power simulation for $\mathsf{MA}_2 \subseteq \mathsf{AM}_1$ follows from the previous results, by carrying out the two steps (swapping the quantifiers and making the error 1-sided) in either order. To prove Theorem 2 we need a single quadratic simulation that handles both steps at the same time. Our proof ends up resembling the proof of $\mathsf{S}_2{\cdot}\mathsf{BPP} \subseteq \mathsf{S}_2\mathsf{P}$ in [RS98], but (similarly to [Dv06]) we start by doing randomness-efficient amplification with very explicit expanders, and we also set parameters differently (in particular, using constant numbers of shifts).

Proof (of Theorem 2). By randomness-efficient amplification [CW89,IZ89] using the expander graph of [GG81], we may assume that Arthur has error probability $<2^{-n}$ while using $O(n)$ random bits and running in time $O(n^2)$. That is, for $L \in \mathsf{MA}_2\text{-}\mathsf{TIME}(n)$ there is a deterministic $O(n^2)$-time algorithm M and a constant c such that if $x \in L$ then $\exists w \in \{0,1\}^{cn}\ \mathbb{P}_{r\in\{0,1\}^{cn}}[M(x,w,r) \text{ accepts}]>1-2^{-n}$, and if $x \notin L$ then $\forall w \in \{0,1\}^{cn}\ \mathbb{P}_{r\in\{0,1\}^{cn}}[M(x,w,r) \text{ accepts}]<2^{-n}$. Consider the $O(n^2)$-time algorithm M' that, letting $a := c^2+c+1$ and $b := c$, interprets its input as $x \in \{0,1\}^n$, $r' := r_1 \cdots r_a \in (\{0,1\}^{cn})^a$, and $w' := ws_1 \cdots s_b \in \{0,1\}^{cn} \times (\{0,1\}^{cn})^b$, and accepts iff $\forall i \in [a]\ \exists j \in [b]\ M(x,w,r_i \oplus s_j)$ accepts. We claim that M' witnesses $L \in \mathsf{AM}_1\text{-}\mathsf{TIME}(n^2)$. First suppose $x \in L$, and fix $w \in \{0,1\}^{cn}$ such that $\mathbb{P}_r[M(x,w,r) \text{ accepts}]>1-2^{-n}$. If we pick $s_1 \cdots s_b \in (\{0,1\}^{cn})^b$ uniformly at random, then for each $r \in \{0,1\}^{cn}$ we have $\mathbb{P}_{s_1\cdots s_b}\big[\neg\exists j\ M(x,w,r\oplus s_j) \text{ accepts}\big]<(2^{-n})^b = 2^{-cn}$. Hence by a union bound, there exists $s_1 \cdots s_b$ such that for all r there exists a j such that $M(x,w,r\oplus s_j)$ accepts. Letting $w' := ws_1 \cdots s_b$, we have $\exists w'\ \forall r'\ M'(x,r',w')$ accepts, and thus $\mathbb{P}_{r'}\big[\exists w'\ M'(x,r',w') \text{ accepts}\big] = 1$. Now suppose $x \notin L$. If we pick $r' \in (\{0,1\}^{cn})^a$ uniformly at random, then for each $w' := ws_1 \cdots s_b$ we have $\mathbb{P}_{r'}\big[\forall i\ \exists j\ M(x,w,r_i\oplus s_j) \text{ accepts}\big]<(b2^{-n})^a \le \frac{1}{2}\cdot 2^{-(cn+bcn)}$. By a union bound, $\mathbb{P}_{r'}\big[\exists w'\ M'(x,r',w') \text{ accepts}\big] \le 1/2$. $\square$

4 Relativized $\mathsf{MA} \not\subseteq \mathsf{NP}^{\mathsf{BPP}}$

The distinction between MA_1 and $\mathsf{N{\cdot}coRP}$ is that when Merlin sends a "wrong" witness for a 1-input, MA_1 allows Arthur to accept with arbitrary probability, whereas $\mathsf{N{\cdot}coRP}$ requires Arthur to accept with a "legal" probability (in $[0, 1/2] \cup \{1\}$). The distinction between MA_2 and $\mathsf{N{\cdot}BPP}$ is similar but where the legal probabilities are $[0, 1/3] \cup [2/3, 1]$. Since the relativizing polynomial-time class equalities $\mathsf{MA} := \mathsf{MA}_2 = \mathsf{MA}_1$ and $\mathsf{NP}^{\mathsf{BPP}} = \mathsf{N{\cdot}BPP} = \mathsf{N{\cdot}coRP}$ hold, the following theorem shows that the distinction is significant.

Theorem 3. *There is an oracle relative to which* $\mathsf{MA} \not\subseteq \mathsf{NP}^{\mathsf{BPP}}$.

Theorem 3 was shown in [FFKL03] using the machinery of generics. In contrast, most oracle separations of pairs of ordinary complexity classes are known to hold directly via separations of the corresponding query complexity (decision tree) models. When we asked Mika Göös whether a query complexity style argument could be used to prove Theorem 3, he promptly manufactured such an argument. He declined coauthorship but graciously gave permission to present the argument for the sake of recording it in the literature. Furthermore, this argument even yields an oracle relative to which $\mathsf{MA} \not\subseteq \mathsf{NP}^{\mathsf{BQP}}$ (by using a quantum rather than randomized query lower bound for the OR function in the appropriate places), which appears to be a new result.

We define a q-query $\mathsf{N{\cdot}BPP}$ decision tree for a partial function $F \colon \{0,1\}^N \to \{0,1\}$ to be a set of probability distributions over depth-q deterministic decision trees, such that for every 0-input, each distribution in the set accepts with probability $\leq 1/3$, and for every 1-input, each distribution in the set accepts with probability in $[0, 1/3] \cup [2/3, 1]$ and at least one of them accepts with probability $\geq 2/3$.[2]

Consider the partial function F_{MA_1} that takes a $2^n \times 2^n$ boolean matrix and evaluates to 1 if there exists an all-1 row, and to 0 if each row has at most half 1's. Theorem 3 is a corollary of the following result.

Lemma 5. *Every* $\mathsf{N{\cdot}BPP}$ *decision tree for* F_{MA_1} *uses* $\Omega(2^n)$ *queries.*

Proof. Let us abbreviate F_{MA_1} as F. Suppose for contradiction there exists an $\mathsf{N{\cdot}BPP}$ decision tree for F using at most 2^{n-4} queries. Define $M^{(0)}$ to be the $2^n \times 2^n$ matrix that has all 1's in its first row and 0's everywhere else. Since $F(M^{(0)}) = 1$, there is a distribution D (which we fix henceforth) in the set of the $\mathsf{N{\cdot}BPP}$ decision tree, that accepts $M^{(0)}$ with probability $\geq 2/3$.

We claim that there exists a sequence of $2^n \times 2^n$ matrices $M^{(0)}, M^{(1)}, \ldots, M^{(2^{n-1})}$ such that for each $i = 1, \ldots, 2^{n-1}$, $M^{(i)}$ has $2^n - i$ 1's in its first row and 0's everywhere else, and the probability D accepts $M^{(i)}$ is within

[2] It would also be natural to charge the log of the size of the set—i.e., the number of nondeterministic guess bits—to the cost of the decision tree. For Theorem 3 it would suffice to consider this more restricted model, but our lower bound holds even for the more powerful model that does not charge for guess bits. [RTVV99] explores this distinction in the context of MA decision trees.

1/8 of the probability D accepts $M^{(i-1)}$. Inductively assuming $M^{(i-1)}$ has been constructed, there must be a 1-entry that gets queried with probability $\leq 2^{n-4}/(2^n - (i-1)) \leq 1/8$ under D, so we can obtain $M^{(i)}$ by flipping this entry to a 0. The claim is proved.

Since $F(M^{(2^{n-1})}) = 0$, D accepts $M^{(2^{n-1})}$ with probability $\leq 1/3$. Hence there exists an i^* such that D accepts $M^{(i^*)}$ with probability within 1/16 of 1/2. This is an illegal probability, but we do not yet have a contradiction, since $M^{(i^*)}$ is not in the domain of F. Now there must be a row of $M^{(i^*)}$ such that the probability (under D) that a bit in that row gets queried is $\leq 2^{n-4}/2^n = 1/16$. Flipping all the 0's to 1's in that row results in a matrix M that D accepts with (illegal) probability within $1/16+1/16$ of 1/2. This is a contradiction since $F(M) = 1$ and so D is supposed to accept M with probability in $[0, 1/3] \cup [2/3, 1]$. $\square$

Acknowledgments. I thank Mika Göös for suggesting the proof of Lemma 5, and anonymous reviewers for helpful comments.

References

[AKR+01] Allender, E., Koucký, M., Ronneburger, D., Roy, S., Vinay, V.: Time-space tradeoffs in the counting hierarchy. In: Proceedings of the 16th Conference on Computational Complexity (CCC), pp. 295–302. IEEE (2001). https://doi.org/10.1109/CCC.2001.933896

[AW09] Aaronson, S., Wigderson, A.: Algebrization: a new barrier in complexity theory. ACM Trans. Comput. Theor. **1**(1), 2:1–2:54 (2009). https://doi.org/10.1145/1490270.1490272

[Bab85] Babai, L.: Trading group theory for randomness. In: Proceedings of the 17th Symposium on Theory of Computing (STOC), pp. 421–429. ACM (1985). https://doi.org/10.1145/22145.22192

[BRS95] Beigel, R., Reingold, N., Spielman, D.: PP is closed under intersection. J. Comput. Syst. Sci. **50**(2), 191–202 (1995). https://doi.org/10.1006/jcss.1995.1017

[Car] Carothers, N.: A short course on approximation theory. Lecture notes. http://personal.bgsu.edu/~carother/Approx.html

[CW89] Cohen, A., Wigderson, A.: Dispersers, deterministic amplification, and weak random sources. In: Proceedings of the 30th Symposium on Foundations of Computer Science (FOCS), pp. 14–19. IEEE (1989). https://doi.org/10.1109/SFCS.1989.63449

[Die07] Diehl, S.: Lower bounds for swapping Arthur and Merlin. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM 2007. LNCS, vol. 4627, pp. 449–463. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_33

[Dv06] Diehl, S., van Melkebeek, D.: Time-space lower bounds for the polynomial-time hierarchy on randomized machines. SIAM J. Comput. **36**(3), 563–594 (2006). https://doi.org/10.1137/050642228

[EZ64] Ehlich, H., Zeller, K.: Schwankung von polynomen zwischen gitterpunkten. Mathematische Zeitschrift **86**, 41–44 (1964)

[FFKL03] Fenner, S., Fortnow, L., Kurtz, S., Li, L.: An oracle builder's toolkit. Inf. Comput. **182**(2), 95–136 (2003). https://doi.org/10.1016/S0890-5401(03)00018-X

[FR99] Fortnow, L., Rogers, J.: Complexity limitations on quantum computation. J. Comput. Syst. Sci. **59**(2), 240–252 (1999). https://doi.org/10.1006/jcss.1999.1651

[GG81] Gabber, O., Galil, Z.: Explicit constructions of linear-sized superconcentrators. J. Comput. Syst. Sci. **22**(3), 407–420 (1981). https://doi.org/10.1016/0022-0000(81)90040-4

[IZ89] Impagliazzo, R., Zuckerman, D.: How to recycle random bits. In: Proceedings of the 30th Symposium on Foundations of Computer Science (FOCS), pp. 248–253. IEEE (1989). https://doi.org/10.1109/SFCS.1989.63486

[KN97] Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)

[Lau83] Lautemann, C.: BPP and the polynomial hierarchy. Inf. Process. Lett. **17**(4), 215–217 (1983). https://doi.org/10.1016/0020-0190(83)90044-3

[RC66] Rivlin, T., Cheney, E.W.: A comparison of uniform approximations on an interval and a finite subset thereof. SIAM J. Numer. Anal. **3**(2), 311–320 (1966)

[Riv81] Rivlin, T.: An Introduction to the Approximation of Functions. Dover, New York City (1981)

[RS98] Russell, A., Sundaram, R.: Symmetric alternation captures BPP. Comput. Complex. **7**(2), 152–162 (1998). https://doi.org/10.1007/s000370050007

[RTVV99] Raz, R., Tardos, G., Verbitsky, O., Vereshchagin, N.: Arthur-Merlin games in Boolean decision trees. J. Comput. Syst. Sci. **59**(2), 346–372 (1999). https://doi.org/10.1006/jcss.1999.1654

[She13] Sherstov, A.: The intersection of two halfspaces has high threshold degree. SIAM J. Comput. **42**(6), 2329–2374 (2013). https://doi.org/10.1137/100785260

[Sip83] Sipser, M.: A complexity theoretic approach to randomness. In: Proceedings of the 15th Symposium on Theory of Computing (STOC), pp. 330–335. ACM (1983). https://doi.org/10.1145/800061.808762

[Ver92] Vereshchagin, N.: On the power of PP. In: Proceedings of the 7th Structure in Complexity Theory Conference (STRUCTURES), pp. 138–143. IEEE (1992). https://doi.org/10.1109/SCT.1992.215389

[Ver95] Vereshchagin, N.: Lower bounds for perceptrons solving some separation problems and oracle separation of AM from PP. In: Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems (ISTCS), pp. 46–51. IEEE (1995). https://doi.org/10.1109/ISTCS.1995.377047

[Vio09] Viola, E.: On approximate majority and probabilistic time. Comput. Complex. **18**(3), 337–375 (2009). https://doi.org/10.1007/s00037-009-0267-3

[vM06] van Melkebeek, D.: A survey of lower bounds for satisfiability and related problems. Found. Trends Theor. Comput. Sci. **2**(3), 197–303 (2006). https://doi.org/10.1561/0400000012

[vW12] van Melkebeek, D., Watson, T.: Time-space efficient simulations of quantum computations. Theor. Comput. **8**(1), 1–51 (2012). https://doi.org/10.4086/toc.2012.v008a001

[Wil08] Williams, R.: Time-space tradeoffs for counting NP solutions modulo integers. Comput. Complex. **17**(2), 179–219 (2008). https://doi.org/10.1007/s00037-008-0248-y

On Counting Perfect Matchings in General Graphs

Daniel Štefankovič[1], Eric Vigoda[2], and John Wilmes[2](✉)

[1] University of Rochester, Rochester, USA
stefanko@cs.rochester.edu
[2] Georgia Institute of Technology, Atlanta, USA
{vigoda,wilmesj}@gatech.edu

Abstract. Counting perfect matchings has played a central role in the theory of counting problems. The permanent, corresponding to bipartite graphs, was shown to be #P-complete to compute exactly by Valiant (1979), and a fully polynomial randomized approximation scheme (FPRAS) was presented by Jerrum, Sinclair, and Vigoda (2004) using a Markov chain Monte Carlo (MCMC) approach. However, it has remained an open question whether there exists an FPRAS for counting perfect matchings in general graphs. In fact, it was unresolved whether the same Markov chain defined by JSV is rapidly mixing in general. In this paper, we show that it is not. We prove torpid mixing for any weighting scheme on hole patterns in the JSV chain. As a first step toward overcoming this obstacle, we introduce a new algorithm for counting matchings based on the Gallai–Edmonds decomposition of a graph, and give an FPRAS for counting matchings in graphs that are sufficiently close to bipartite. In particular, we obtain a fixed-parameter tractable algorithm for counting matchings in general graphs, parameterized by the greatest "order" of a factor-critical subgraph.

1 Introduction

Counting perfect matchings is a fundamental problem in the area of counting/sampling problems. For an undirected graph $G = (V, E)$, let $\mathcal{P}$ denote the set of perfect matchings of G. Can we compute (or estimate) $|\mathcal{P}|$ in time polynomial in $n = |V|$? For which classes of graphs?

A polynomial-time algorithm for the corresponding decision and optimization problems of determining if a given graph contains a perfect matching or finding a matching of maximum size was presented by Edmonds [2]. For the counting problem, a classical algorithm of Kasteleyn [9] gives a polynomial-time algorithm for exactly computing $|\mathcal{P}|$ for planar graphs.

For bipartite graphs, computing $|\mathcal{P}|$ is equivalent to computing the permanent of $n \times n$ $(0, 1)$-matrices. Valiant [14] proved that the $(0, 1)$-Permanent is #P-complete. Subsequently attention turned to the Markov Chain Monte Carlo (MCMC) approach. A Markov chain where the mixing time is polynomial in n is said to be *rapidly mixing*, and one where the mixing time is exponential in

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 873–885, 2018.
https://doi.org/10.1007/978-3-319-77404-6_63

$\Omega(n)$ is referred to as *torpidly mixing*. A rapidly mixing chain yields an FPRAS (fully polynomial-time randomized approximation scheme) for the corresponding counting problem of estimating $|\mathcal{P}|$ [8].

For dense graphs, defined as those with minimum degree >n/2, Jerrum and Sinclair [6] proved rapid mixing of a Markov chain defined by Broder [1], which yielded an FPRAS for estimating $|\mathcal{P}|$. The Broder chain walks on the collection $\Omega = \mathcal{P} \cup \mathcal{N}$ of perfect matchings $\mathcal{P}$ and near-perfect matchings $\mathcal{N}$; a near-perfect matching is a matching with exactly 2 holes or unmatched vertices. Jerrum and Sinclair [6], more generally, proved rapid mixing when the number of perfect matchings is within a $\mathsf{poly}(n)$ factor of the number of near-perfect matchings, i.e., $|\mathcal{P}|/|\mathcal{N}| \geq 1/\mathsf{poly}(n)$. A simple example, referred to as a "chain of boxes" which is illustrated in Fig. 1, shows that the Broder chain is torpidly mixing. This example was a useful testbed for catalyzing new approaches to solving the general permanent problem.

Jerrum et al. [7] presented a new Markov chain on $\Omega = \mathcal{P} \cup \mathcal{N}$ with a non-trivial weighting scheme on the matchings based on the holes (unmatched vertices). They proved rapid mixing for any bipartite graph with the requisite weights used in the Markov chain, and they presented a polynomial-time algorithm to learn these weights. This yielded an FPRAS for estimating $|\mathcal{P}|$ for all bipartite graphs. That is the current state of the art (at least for polynomial-time, or even sub-exponential-time algorithms).

Could the JSV-Markov chain be rapid mixing on non-bipartite graphs? Previously there was no example for which torpid mixing was established, it was simply the case that the proof in [7] fails. We present a relatively simple example where the JSV-Markov chain fails for the weighting scheme considered in [7]. More generally, the JSV-chain is torpidly mixing on our class of examples for any weighting scheme based on the hole patterns, see Theorem 3 in Sect. 2 for a formal statement following the precise definition of the JSV-chain.

A natural approach for non-bipartite graphs is to consider Markov chains that exploit odd cycles or blossoms in the manner of Edmonds' algorithm. We observe that a Markov chain which considers *all* blossoms for its transitions is intractable since sampling all blossoms is NP-hard, see Theorem 5. On the other hand, a chain restricted to minimum blossoms is not powerful enough to overcome our torpid mixing examples. See Sect. 3 for a discussion.

Finally we utilize the Gallai−Edmonds graph decomposition into factor-critical graphs [2–4,12] to present new algorithmic insights that may overcome the obstacles in our classes of counter-examples. In Sect. 4, we describe how the Gallai−Edmonds decomposition can be used to efficiently estimate $|\mathcal{P}|$, the number of perfect matchings, in graphs whose factor-critical subgraphs have bounded order (Theorem 7), as well as in the torpid mixing example graphs (Theorem 8).

Although all graphs are explicitly defined in the text below, figures depicting these graphs are deferred to the appendix.

1.1 Markov Chains

Consider an ergodic Markov chain with transition matrix P on a finite state space Ω, and let π denote the unique stationary distribution. We will usually assume the Markov chain is time reversible, i.e., that it satisfies the **detailed balance condition** $\pi(x)P(x,y) = \pi(y)P(x,y)$ for all states $x, y \in \Omega$.

For a pair of distributions μ and ν on Ω we denote their total variation distance as $d_{\mathsf{TV}}(\mu,\nu) = \frac{1}{2}\sum_{x\in\Omega}|\mu(x) - \nu(x)|$. The standard notion of **mixing time** T_{mix} is the number of steps from the worst starting state $X_0 = i$ to reach total variation distance $\leq 1/4$ of the stationary distribution π, i.e., we write $T_{\text{mix}} = \max_{i\in\Omega}\min\{t : d_{\mathsf{TV}}(P^t(i,\cdot),\pi) \leq 1/4\}$.

We use conductance to obtain lower bounds on the mixing time. For a set $S \subset \Omega$ its **conductance** is defined as:

$$\Phi(S) = \frac{\sum_{x\in S, y\notin S}\pi(x)P(x,y)}{\sum_{x\in S}\pi(x)}.$$

Let $\Phi_* = \min_{S\subset\Omega:\pi(S)\leq 1/2}\Phi(S)$. Then (see, e.g., [10,13])

$$T_{\text{mix}} \geq \frac{1}{4\Phi_*}. \tag{1}$$

1.2 Factor-Critical Graphs

A graph $G = (V,E)$ is **factor-critical** if for every vertex $v \in V$, the graph induced on $V \setminus \{v\}$ has a perfect matching. (In particular, $|V|$ is odd.)

Factor-critical graphs are characterized by their "ear" structure. The **quotient** G/H of a graph G by a (not necessarily induced) subgraph H is derived from G by deleting all edges in H and contracting all vertices in H to a single vertex v_H (possibly creating loops or multi-edges). An **ear** of G relative a subgraph H of G is simply a cycle in G/H containing the vertex v_H.

Theorem 1 (Lovász [11]). *A graph G is factor-critical if and only if there is a decomposition $G = C_0 \cup \cdots \cup C_r$ such that C_0 is a single vertex, and C_i is an odd-length ear in G relative to $\bigcup_{j<i} C_j$, for all $0 < i \leq r$.*

Furthermore, if G is factor critical, there exists such a decomposition for every choice of vertex C_0, and the order *r of the decomposition is independent of all choices.*

Since the number of ears in the ear decomposition of a factor-critical graph depends only on the graph, and not on the choice made in the decomposition, we say the **order** of the factor-critical graph G is the number r of ears in any ear decomposition of G.

Factor-critical graphs play a central role in the Gallai–Edmonds structure theorem for graphs. We state an abridged version of the theorem below.

Given a graph G, let $D(G)$ be the set of vertices that remain unmatched in at least one maximum matching of G. Let $A(G)$ be the set of vertices not in $D(G)$ but adjacent to at least one vertex of $D(G)$. And let $C(G)$ denote the remaining vertices of G.

Theorem 2 (Gallai–Edmonds Structure Theorem). *The connected components of $D(G)$ are factor-critical. Furthermore, every maximum matching of G induces a perfect matching on $C(G)$, a near-perfect matching on each connected component of $D(G)$, and matches all vertices in $A(G)$ with vertices from distinct connected components of $D(G)$.*

2 The Jerrum–Sinclair–Vigoda Chain

We recall the definition of the original Markov chain proposed by Broder [1]. The state space of the chain is $\Omega = \mathcal{P} \cup \bigcup_{u,v} \mathcal{N}(u,v)$ where $\mathcal{P}$ is the collection of perfect matchings and $\mathcal{N}(u,v)$ are near-perfect matchings with holes at u and v (i.e., vertices u and v are the only unmatched vertices). The transition rule for a matching $M \in \Omega$ is as follows:

1. If $M \in \mathcal{P}$, randomly choose an edge $e \in M$ and transition to $M \setminus \{e\}$.
2. If $M \in \mathcal{N}(u,v)$, randomly choose a vertex $x \in V$. If $x \in \{u,v\}$ and u is adjacent to v, transition to $M \cup \{(u,v)\}$. Otherwise, let $y \in V$ be the vertex matched with x in M, and randomly choose $w \in \{u,v\}$. If x is adjacent to w, transition to the matching $M \cup \{(x,w)\} \setminus \{(x,y)\}$.

The chain $\mathfrak{X}_{\mathrm{B}}$ is symmetric, so its stationary distribution is uniform. In particular, when $|\mathcal{P}|/|\Omega|$ is at least inverse-polynomial in n, we can efficiently generate uniform samples from $\mathcal{P}$ via rejection sampling, given access to samples from the stationary distribution of $\mathfrak{X}_{\mathrm{B}}$.

In order to sample perfect matchings even when $|\Omega|/|\mathcal{P}|$ is exponentially large, Jerrum et al. [7] introduce a new chain $\mathfrak{X}_{\mathrm{JSV}}$ that changes the stationary distribution of $\mathfrak{X}_{\mathrm{B}}$ by means of a Metropolis filter. The new stationary distribution is uniform across *hole patterns*, and then uniform within each hole pattern, i.e., for every $M \in \Omega$, the stationary probability of M is proportional to $1/|\mathcal{N}(u,v)|$ if $M \in \mathcal{N}(u,v)$, and proportional to $1/|\mathcal{P}|$ if $M \in \mathcal{P}$.

We define $\mathfrak{X}_{\mathrm{JSV}}$ in greater detail. For $M \in \Omega$, define the weight function

$$w(M) = \begin{cases} \frac{1}{|\mathcal{P}|} & \text{if } M \in \mathcal{P} \\ \frac{1}{|\mathcal{N}(u,v)|} & \text{if } M \in \mathcal{N}(u,v) \end{cases} \tag{2}$$

Definition 1. *The chain $\mathfrak{X}_{JSV}$ has the same state space as $\mathfrak{X}_B$. The transition rule for a matching $M \in \Omega$ is as follows:*

1. *First, choose a matching $M' \in \Omega$ to which M may transition, according to the transition rule for $\mathfrak{X}_B$.*
2. *With probability $\min\{1, w(M')/w(M)\}$, transition to M'. Otherwise, stay at M.*

In their paper, Jerrum et al. [7] in fact analyze a more general version of the chain $\mathfrak{X}_{\mathrm{JSV}}$ that allows for arbitrary edge weights in the graph. They show that the chain is rapidly mixing for bipartite graphs G. (They also study the separate problem of estimating the weight function w, and give a "simulating annealing" algorithm that allows the weight function w to be estimated by gradually adjusting edge weights to obtain an arbitrary bipartite graph G from the complete bipartite graph.) Their analysis of the mixing time uses a canonical paths argument that crucially relies on the bipartite structure of the graph. However, it remained an open question whether a different analysis of the same chain $\mathfrak{X}_{\mathrm{JSV}}$, perhaps using different canonical paths, might generalize to non-bipartite graphs. We rule out this approach.

In fact, we rule out a more general family of Markov chains for sampling perfect matchings. We say a Markov chain is "of $\mathfrak{X}_{\mathrm{JSV}}$ type" if it has the same state space as $\mathfrak{X}_{\mathrm{JSV}}$, with transitions as defined in Definition 1, for *some* weight function $w(M)$ (not necessarily the same as in Eq. (2)) depending only the hole pattern of the matching M.

Theorem 3. *There exists a graph G on n vertices such that for any Markov chain $\mathfrak{X}$ of $\mathfrak{X}_{JSV}$ type on G, either the stationary probability of $\mathcal{P}$ is at most* $\exp(-\Omega(n))$*, or the mixing time of $\mathfrak{X}$ is at least* $\exp(\Omega(n))$.

The graph G of Theorem 3 is constructed from several copies of a smaller gadget H, which we now define.

Definition 2. *The **chain of boxes gadget** B_k of length k is the graph on $4k$ vertices depicted in Fig. 1. To construct B_k, we start with a path $P_{2k-1} = v_0, v_1, \ldots, v_{2k}$ of length $2k-1$. Then, for every even edge $\{v_{2i}, v_{2i+1}\}$ on the path, we add two additional vertices a_i, b_i, along with edges to form a path $v_{2i}, a_i, b_i, v_{2i+1}$ of length 3.*

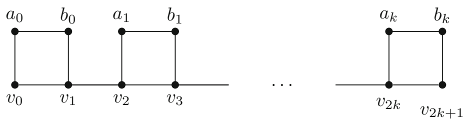


Fig. 1. The "chain of boxes" gadget B_k, which has 2^k perfect matchings, but only a single matching in $\mathcal{N}(v_0, v_{2k+1})$.

Observation 4. *The chain of boxes gadget B_k has 2^k perfect matchings, but only one matching in $\mathcal{N}(v_0, v_{2k+1})$.*

Definition 3. *The **torpid mixing gadget** H_k is the graph depicted in Fig. 2. To construct H, first take a C_{12} and label two antipodal vertices as a and b. Add an edge between a and b, and label the two vertices farthest from a and b as u and v. Label the neighbor of u closest to a as w_1, and the other neighbor of u as w_2. Label the neighbor of v closest to a as z_1 and the other neighbor of v as z_2. Finally, add four chain-of-boxes gadgets B_k, identifying the vertices v_0 and v_{2k} of the gadgets with w_1 and a, with a and z_1, with w_2 and b, and with b and z_2, respectively.*

Note that in Figs. 2 and 3, one "box" from each copy of B_k in the torpid mixing gadget is left undrawn, for visual clarity.

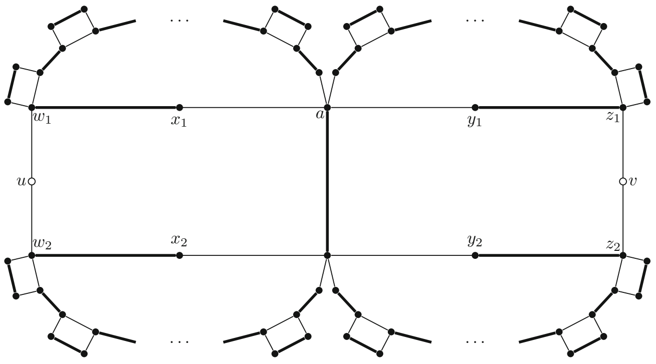


Fig. 2. The torpid mixing gadget H_k. The unique matching $M \in \mathcal{N}(u, v)$ is depicted with thick edges.

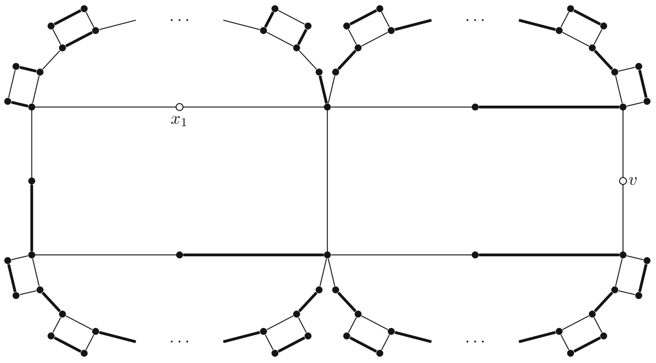


Fig. 3. A matching $M' \in \mathcal{N}(x_1, v)$. There are exponentially many matchings with the same hole pattern, obtained by alternating the 4-cycles above x_1.

Lemma 1. *The torpid mixing gadget* $H = H_k$ *has* $16k + 4$ *vertices and exactly* 2 *perfect matchings. Furthermore,* $|\mathcal{N}_H(u, v)| = 1$ *and* $\mathcal{N}_H(x_1, v) \geq 2^k$.

The unique matching in $\mathcal{N}_H(u, v)$ is depicted in Fig. 2, and an example of a matching in $\mathcal{N}_H(x_1, v)$, which generalizes easily to the entire family, is depicted in Fig. 3. The details of the proof are deferred to the full version of the paper.

The torpid mixing gadget already suffices on its own to show that the Markov chain $\mathfrak{X}_{\mathfrak{X}_{\mathrm{JSV}}}$ defined in [7] is torpidly mixing. In particular, the conductance out of the set $\mathcal{N}_H(x_1, v) \subseteq \Omega(H)$ is $2^{-\Omega(k)}$. In order to prove the stronger claim of Theorem 3, that every Markov chain of $\mathfrak{X}_{\mathrm{JSV}}$-type fails to efficiently sample perfect matchings, we construct a slightly larger graph from copies of the torpid mixing gadgets.

Definition 4. *The* ***counterexample graph*** G_k *is the graph depicted in Fig. 4. It is defined by replacing every third edge of the twelve-cycle* C_{12} *with the gadget* H_k *defined in Fig. 2. Specifically, let* $\{u_i, v_i\}$ *be the* $3i$*-th edge of* C_{12} *for* $i \in \{1, \ldots, 4\}$*. We delete each edge* $\{u_i, v_i\}$ *and replace it with a copy of* H*, identifying the vertices* u *and* v *of* H *with vertices* u_i *and* v_i *of* C_{12}*. The resulting graph is* G_k*. Thus, of the* 12 *original vertices in* C_{12}*,* 8 *of the corresponding vertices in* G_k *participate in a copy of the gadget* H*, and* 4 *do not. These* 4 *vertices of* G_k *which do not participate in any copy of the gadget* H *are labeled* $t_1, \ldots, t_4$ *in cyclic order, and the copies of the gadget* H *are labeled* $H_1, \ldots H_4$ *in cyclic order, with* H_1 *coming between* t_1 *and* t_2*, and so on. Thus,* t_1 *is adjacent to* u_1 *and* v_4*,* t_i *is adjacent to* u_i *and* v_{i-1} *for* $i \in \{2, \ldots, 4\}$*, and* H_i *contains both* u_i *and* v_i*.*

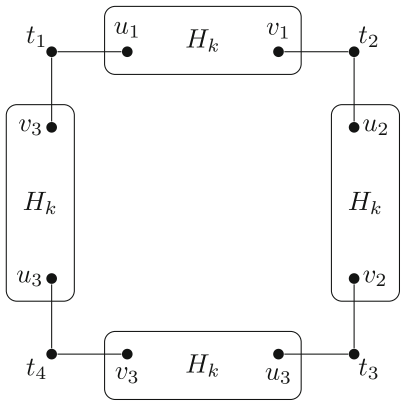


Fig. 4. The "counterexample graph" G_k on which $\mathfrak{X}_{\text{JSV}}$ is torpidly mixing. The boxes labeled H_k represent copies of the torpid mixing gadget of Definition 3.

In particular, G_k has $4|V(H)| + 4 = 64k + 8$ vertices.

The perfect and near-perfect matchings of G_k are naturally divided into four intersecting families. For $i \in \{1, \ldots, 4\}$ we define S_i to be the collection of (perfect and near-perfect) matchings $M \in \Omega(G_k)$ such that the restriction of M to H_i has two holes, at u_i and v_i, i.e., such that the vertices u_i and v_i either have holes in M or are matched outside of H_i.

Lemma 2. *The counterexample graph* G_k *has exactly* 8 *perfect matchings. Of these,* 4 *are in* $S_1 \cap S_3 \setminus (S_2 \cup S_4)$ *and* 4 *are in* $S_2 \cap S_4 \setminus (S_1 \cup S_3)$*.*

The proof of this lemma is deferred to the full version of the paper.

In the proof below, we use the notation $\mathcal{N}(M)$ denote the collection of matchings with the hole pattern as M. That is, $\mathcal{N}(M) = \mathcal{P}$ if $M \in \mathcal{P}$, and $\mathcal{N}(M) = \mathcal{N}(u, v)$ if $M \in \mathcal{N}(u, v)$.

Proof (Proof of Theorem 3). Let G_k be the counterexample graph of Definition 4. We will show that the set $S_1 \cup S_3 \subseteq \Omega(G_k)$ has poor conductance, unless the stationary probability of $\mathcal{P}_{G_k}$ is small. We will write $A = S_1 \cup S_3$ and $\overline{A} = \Omega(G_k) \setminus (S_1 \cup S_3)$.

Let $M \in A$ and $M' \in \overline{A}$ be such that $P(M, M') > 0$. We claim that neither M nor M' are perfect matchings. Assume without loss of generality that $M \in S_1$. If $M \in S_1$ is a perfect matching, then $M \in P_2$ and so $M \in S_3$. The only legal transitions from M to $\Omega \setminus S_1$ are those that introduce additional holes within H_1, but none of these transitions to a matching outside of S_3. Hence, M cannot be perfect. But if M' is perfect, then $M' \in P_1$, and so M' induces a perfect matching on S_1. But then the transition from M to M' must simultaneously affect u_1 and v_1, and no such transition exists.

We denote by $\partial\overline{A}$ the set of matchings $M' \in \overline{A}$ such that there exists a matching $M \in A$ with $P(M, M') > 0$. We claim that for every matching $M' \in \overline{A}$, we have

$$|\mathcal{N}(M') \cap \partial\overline{A}| \leq 2^{k-1}|\mathcal{N}(M')|\,. \tag{3}$$

Let $M' \in \partial\overline{A}$, and let $M \in A$ be such that $P(M, M') > 0$. Suppose first that $M \in S_1$. Label the vertices of H_1 as in Fig. 2, identifying u_1 with u and v_1 with v. Let N be the matching on $H = H_1$ induced by M, and let N' be the matching on H_1 induced by M'. We have $N \in \mathcal{N}_H(u_1, v_1)$. But by Lemma 1, we have $|\mathcal{N}_H(u_1, v_1)| = 1$, i.e., N is exactly the matching depicted in Fig. 2. The only transitions that remove the hole at u are the two that shift the hole to x_1 or x_2, and the only transitions that remove the hole at v are the two that shift the hole to y_1 or y_2. So, without loss of generality, by the symmetry of G_k, we have $N' \in \mathcal{N}_H(x_1, v_1)$. By Lemma 1, $|\mathcal{N}_H(x_1, v_1)| \geq 2^k$, but only one matching in $\mathcal{N}_H(x_1, v_1)$ has a legal transition to N. Therefore, if we replace the restriction of M' to H_1 with any other matching in $\mathcal{N}_H(x_1, v_1)$, we obtain another matching $M'' \in \mathcal{N}(M')$, but M'' has no legal transition to any matching in $\mathcal{N}(M)$. Hence, only a 2^{-k}-fraction of $\mathcal{N}(M')$ has a legal transition to S_1, and similarly only a 2^{-k}-fraction of $\mathcal{N}(M')$ has a legal transition to S_3. In particular, we have proved Eq. (3).

From Eq. (3), it immediately follows that the stationary probability of $\partial\overline{A}$ is

$$\pi(\partial\overline{A}) = \sum_{M' \in \partial\overline{A}} \pi(M') = \sum_{M' \in \overline{A}} \pi(M') \frac{|\mathcal{N}(M') \cap \partial\overline{A}|}{|\mathcal{N}(M')|} = 2^{-k+1}\pi(\overline{A}) \tag{4}$$

We now compute

$$\sum_{\substack{M \in A, M' \in \overline{A} \\ P(M,M') > 0}} \pi(M)P(M, M') = \sum_{\substack{M \in A, M' \in \overline{A} \\ P(M,M') > 0}} \pi(M')P(M', M) \leq \pi(\partial(\overline{A}))$$
$$< 2^{-k+1}\pi(\overline{A}),$$

where we first use the detailed balance condition and then Eq. (4).

Now by (1) and the definition of conductance, we have

$$\frac{1}{4\tau_{\mathfrak{X}}} < \varPhi(A) < 2^{-k}\frac{\pi(\overline{A})}{\pi(A)}\,.$$

In particular, if $\tau_{\mathfrak{X}} < 2^{k/2-2}$, then $\pi(\overline{A}) > 2^{k/2+1}\pi(A)$. Suppose this is the case. By Lemma 2, half of the perfect matchings of G_k belong to A. In particular, $\pi(\mathcal{P}_{G_k}) \leq 2\pi(A) < 2^{-k/2+2}$. Hence, either the stationary probability of $\mathcal{P}$ is at most $2^{-k/2+2} = \exp(-\varOmega(n))$, or the mixing time of $\mathfrak{X}$ is at least $2^{k/2-2} = \exp(\varOmega(n))$. □

We remark that the earlier Markov chain studied by Broder [1] and Jerrum and Sinclair [6] is also torpidly mixing on the counterexample graph of Definition 4, since the ratio of near-perfect matchings to perfect matchings is exponential [6].

3 Chains Based on Edmonds' Algorithm

Given that Edmonds' classical algorithm for *finding* a perfect matching in a bipartite graph requires the careful consideration of odd cycles in the graph, it is reasonable to ask whether a Markov chain for counting perfect matchings should also somehow track odd cycles. In this section, we briefly outline some of the difficulties of such an approach.

A *blossom* of length k in a graph G equipped with a matching M is simply an odd cycle of length $2k+1$ in which k of the edges belong to M. Edmonds' algorithm finds augmenting paths in a graph by exploring the alternating tree rooted at an unmatched vertex, and contracting blossoms to a vertex as they are encountered. Given a blossom B containing an unmatched vertex u, there is an alternating path of even length to every vertex $v \in B$. *Rotating* B to v means shifting the hole at u to v by alternating the u-v path in B.

Adding rotation moves to a Markov chain in the style of $\mathfrak{X}_{\mathrm{JSV}}$ is an attractive possible solution to the obstacles presented in the previous section. Indeed, if it were possible to rotate the 7-cycle containing u and a in the graph in Fig. 2, it might be possible to completely avoid problematic holes at x_1 or x_2.

The difficulty in introducing such an additional move the Markov chain $\mathfrak{X}_{\mathrm{JSV}}$ is in defining the set of *feasible* blossoms that may be rotated, along with a probability distribution over such blossoms. In order to be useful, we must be able to *efficiently sample* from the feasible blossoms at a given near-perfect matching M. Furthermore, the feasible blossoms must respect time reversibility: if B is feasible when the hole is at $u \in B$, then it must also be feasible after rotating the hole to $v \in B$; reversibility of the Markov chain is needed so that we understand its stationary distribution. Finally, the feasible blossoms must be rich enough to avoid the obstacles outlined in the previous section.

The set of "minimum length" blossoms at a given hole vertex u satisfies the first criterion of having an efficient sampling algorithm. But it is easy to see that if only minimum length blossoms are feasible, then the obstacles outlined in the previous section will still apply (simply by adding a 3-cycle at every vertex). Moreover, families blossoms characterized by minimality may struggle to satisfy the second criterion of time-reversibility. In Fig. 5, there is a unique blossom containing u, but after rotating the hole to v, it is no longer minimal.

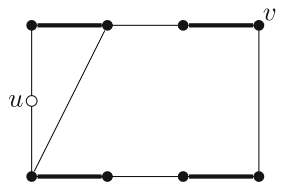


Fig. 5. After rotating the blossom so that the hole is moved from u to v, the blossom is no longer "minimal".

On the other hand, the necessity of having an efficient sampling algorithm for the feasible blossoms already rules out the simplest possibility, namely, the uniform distribution over *all* blossoms containing a given hole vertex u. Indeed, if we could efficiently sample from the uniform distribution over all blossoms containing a given vertex u, then by an entropy argument we could find arbitrarily large odd cycles in the graph, which is NP-hard.

Theorem 5. *Let* Sampling Blossoms *problem be defined as follows. The input is an undirected graph G and a near-perfect matching M with holes at $w, r \in V(G)$. The output is a uniform sample from the uniform distribution of blossoms containing w. Unless NP=RP there is no randomized polynomial-time sampler for* Sampling Blossoms.

The proof is deferred to the full version of the paper.

4 A Recursive Algorithm

We now explore a new recursive algorithm for counting matchings, based on the Gallai−Edmonds decomposition. In the worst case, this algorithm may still require exponential time. However, for graphs that have additional structural properties, for example, those that are "sufficiently close to bipartite" in a sense that will be made precise, our recursive algorithm runs in polynomial time. In particular, it will run efficiently on examples similar to those used to prove torpid mixing of Markov chains in the previous section.

We now state the algorithm. It requires as a subroutine an algorithm for computing the permanent of the bipartite adjacency matrix of a bipartite graph G up to accuracy ε. We denote this subroutine by Permanent(G, ε). The Permanent subroutine requires time polynomial in $|V(G)|$ and $1/\varepsilon$ using the algorithm of Jerrum et al. [7], but we use it as a black-box.

We first argue the correctness of the algorithm.

Algorithm 1. Recursive algorithm for approximately counting the number of perfect matchings in a graph

```
1: procedure Recursive-Count($G, \varepsilon$)
2:     If $V(G) = \emptyset$, return 1.
3:     Choose $u \in V(G)$.
4:     Compute the Gallai–Edmonds decomposition of $G - u$.
5:     for all $v \in D(G - u)$ do
6:         $H_v \leftarrow$ the connected component of $G - u$ containing $v$
7:         $m_v \leftarrow$ Recursive-Count($H_v - v, \varepsilon/(2n)$)
8:     end for
9:     $m_C \leftarrow$ Recursive-Count($C(G - u), \varepsilon/3$)
10:    Let $X = A(G - u) \cup \{u\}$, and let $Y$ be the set of connected components in
       $D(G - u)$. Let $G'$ be the bipartite graph on $(X, Y)$ defined as follows: for every
       $x \in X$ and $H \in Y$, if $x$ has any neighbors in $H$ in $G'$, add an edge $\{x, H\}$ in $G'$
       with weight
           $w(x, H) = \sum_{v \in N(x) \cap H} m_v$ .
11:    return $m_C *$ Permanent($G', \varepsilon/3$)
12: end procedure
```

Theorem 6. *Algorithm 1 computes the number of perfect matchings in G to within accuracy ε.*

Proof. We show that the algorithm is correct for graphs on n vertices, assuming it is correct for all graphs on at most $n - 1$ vertices.

We claim that permanent of the incidence matrix of G' defined on line 10 equals the number of perfect matchings in G. Indeed, every perfect matching M of G induces a maximum matching M_u on $G - u$. By the Gallai–Edmonds theorem, M_u matches each element of $A(G')$ with a vertex from a distinct component of $D(G')$, leaving one component of $D(G')$ unmatched. Vertex u must therefore be matched in M with a vertex from the remaining component of $D(G')$. Therefore, M induces a perfect matching M' on G'. Now let $H_x \in Y$ be the vertex of G' matched to x for each $x \in X$. Then the number of distinct matchings of G inducing the same matching M'' on G'' is exactly

$$\prod_{x \in X} \sum_{v \in N(x) \cap H_x} m_v = \prod_{x \in X} w(x, H_x)$$

which is the contribution of M' to the permanent of G'. Similarly, from an arbitrary matching M' of G', with H_x defined as above, we obtain $\prod_{x \in X} w(x, H_x)$ matchings of G, proving the claim.

Hence, it suffices to to compute the permanent of the incidence matrix of G' up to accuracy ε. We know the entries of the incidence matrix up to accuracy $\varepsilon/(2n)$, and $(1+\varepsilon/(2n))^{n/2} < 1+\varepsilon/3$ for ε sufficiently small. Therefore, it suffices to compute the permanent of our approximation of the incidence matrix up to accuracy $\varepsilon/3$ to get overall accuracy better than ε. □

The running time of Algorithm 1 is sensitive to the choice of vertex u on line 3. If u can be chosen so that each component of $D(G-u)$ is small, then the algorithm is an efficient divide-and-conquer strategy. More generally, if u can be chosen so that each component of $D(G-u)$ is in some sense "tractable", then an efficient divide-and-conquer strategy results. In particular, since it is possible to exactly count the number of perfect matchings in a factor-critical graph of bounded order in polynomial time, we obtain an efficient algorithm for approximately counting matchings in graphs whose factor-critical subgraphs have bounded order. This is the sense in which Algorithm 1 is efficient for graphs "sufficiently close" to bipartite.

Theorem 7. *Suppose every factor-critical subgraph of G has order at most k. Then the number of perfect matchings in G can be counted to within accuracy ε in time $2^{O(k)}\mathsf{poly}(n, 1/\varepsilon)$.*

The essential idea of the proof is to first observe that a factor-critical graph can be shrunk to a graph with $O(k)$ edges having the same number of perfect matchings after deleting any vertex. The number of perfect matchings can then be counted by brute force in time $2^{O(k)}\mathsf{poly}(n)$. This procedure replaces the recursive calls on line 6 of the algorithm. The details of the proof are deferred to the full version of the paper.

We note that Theorem 7 is proved by eliminating recursive calls in the algorithm. Although the recursive calls of Algorithm 1 can be difficult to analyze, they can also be useful, as we now demonstrate by showing that Algorithm 1 runs as-is in polynomial time on the counterexample graph of Definition 4, for appropriate choice of the vertex u on the line 3 of the algorithm.

Theorem 8. *Algorithm 1 runs in polynomial time on the counterexample graph of Definition 4, for appropriate choice of the vertex u on the line 3 of the algorithm.*

The proof is given in the full version of the paper.

Acknowledgements. This research was supported in part by NSF grants CCF-1617306, CCF-1563838, CCF-1318374, and CCF-1717349. The authors are grateful to Santosh Vempala for many illuminating conversations about Markov chains and the structure of factor-critical graphs.

References

1. Broder, A.Z.: How hard is it to marry at random? (On the approximation of the permanent). In: Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC), pp. 50–58 (1986). Erratum in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, p. 551 (1988)
2. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
3. Gallai, T.: Kritische Graphen II. Magyar Tud. Akad. Mat. Kutató Int. Kőzl. **8**, 273–395 (1963)

4. Gallai, T.: Maximale systeme unabhängiger kanten. Magyar Tud. Akad. Mat. Kutató Int. Kőzl **9**, 401–413 (1964)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
6. Jerrum, M., Sinclair, A.: Approximating the permanent. SIAM J. Comput. **18**(6), 1149–1178 (1989)
7. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. J. ACM **51**(4), 671–697 (2004)
8. Jerrum, M.R., Valiant, L.G., Vazirani, V.V.: Random generation of combinatorial structures from a uniform distribution. Theoret. Comput. Sci. **43**(2–3), 169–188 (1986)
9. Kasteleyn, P.W.: Graph theory and crystal physics. In: Graph Theory and Theoretical Physics, pp. 43–110, Academic Press, London (1967)
10. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing Times. American Mathematical Society, Providence (2009)
11. Lovász, L.: A note on factor-critical graphs. Stud. Sci. Math. Hungar **7**(11), pp. 279–280 (1972)
12. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Hoboken (1998)
13. Sinclair, A.J.: Algorithms for Random Generation and Counting: A Markov Chain Approach. Birkhäuser, Basel (1988)
14. Valiant, L.G.: The complexity of computing the permanent. Theoret. Comput. Sci. **8**(2), 189–201 (1979)

Author Index